21800

National Library of Canada | Bibliothèque nationale du Canada

CANADIAN THESES ON MICROFICHE

THÈSES CANADIENNES SUR MICROFICHE

NAME OF AUTHOR/*NOM DE L'AUTEUR* GORDON Francis Paul Deecker

TITLE OF THESIS/*TITRE DE LA THÈSE* Interactive Graphics and a Planning Problem

UNIVERSITY/*UNIVERSITÉ* University of Alberta

DEGREE FOR WHICH THESIS WAS PRESENTED/ *GRADE POUR LEQUEL CETTE THÈSE FUT PRÉSENTÉE* Phd

YEAR THIS DEGREE CONFERRED/*ANNÉE D'OBTENTION DE CE DEGRÉ* 1974

NAME OF SUPERVISOR/*NOM DU DIRECTEUR DE THÈSE* Dr. T. A. Marsland

Permission is hereby granted to the NATIONAL LIBRARY OF CANADA to microfilm this thesis and to lend or sell copies of the film.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

L'autorisation est, par la présente, accordée à la BIBLIOTHÈQUE NATIONALE DU CANADA de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans l'autorisation écrite de l'auteur.

DATED/*DATÉ* Oct 24 1974 SIGNED/*SIGNÉ* Gordon Deecker

PERMANENT ADDRESS/*RÉSIDENCE FIXÉ*
Dept of Computer Science
University of Canterbury
Christchurch 1 New Zealand

NL-91 (11-73)

THE UNIVERSITY OF ALBERTA

INTERACTIVE GRAPHICS AND A PLANNING PROBLEM

by

© Gordon Francis Deecker

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH
IN PARTIAL FULFILMENT OF THE REQUIREMENTS FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTING SCIENCE

EDMONTON, ALBERTA
FALL, 1974

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled "INTERACTIVE GRAPHICS AND A PLANNING PROBLEM", submitted by GORDON FRANCIS DEECKER in partial fulfilment of the requirements for the degree of Doctor of Philosophy.

T.A. Marsland
Supervisor

W.A. Davis

W.G. Adams

E.R. Reinelt

Ronald Baecker
External Examiner

Date October 3, 1974

ABSTRACT

An investigation into the use of interactive graphics for campus planning is described in this thesis. An interactive graphics implementation has been developed for the Sieve Process, a planning procedure for determining the land-site of a building.

While designed as an implementation of the Sieve Process, the system is effectively a general-purpose interactive map storage and retrieval system. Its design and implementation revealed a number of problems common to many map storage and retrieval applications. Two such problems, the question of choosing from alternative digital encoding methods, and the design of a structured data base, have been studied.

Storage requirements for three possible digital representations of map data were considered. The results obtained show important differences from those published elsewhere. It was found that the relative merits for storage economy are effected by the magnitude of the allowable error in relation to the scale.

In order to transfer to the computer some of the tasks for evaluation of alternative land-sites, a key problem is to define a data structure which contains information regarding campus resources and other data required for evaluation purposes. A computer model of campus resources, including a data base and the necessary support routines, has been developed for use in an interactive environment.

The considerable time needed to develop the map storage and retrieval application led to the study of two major problems common in all interactive graphics programs: the definition of a Console Command Language (CCL), and the implementation of an interpreter for such a language.

A new graphics interface between the user at the CRT and the application program has been developed. The definition of elements in the CCL has been formalized. A CCL interpreter, for use with any interactive graphics program, has been implemented. A major improvement is that the user programming requirements have been modularized, that is, a user may program modules for each command in the CCL independently.

ACKNOWLEDGEMENTS

Sincere thanks are due to Professor J. P. Penny, for his advice and guidance throughout the duration of this research project.

I wish to express thanks also to Dr. T. A. Marsland, who took over the role of supervisor during the latter stages, to R. C. Enerson, for his many discussions during the formative stages of the work, and to Dr. R. M. Baecker, for the suggestions for improvements which have been incorporated into this thesis.

Finally, I owe thanks to my wife, Dorothy, for the long hours spent typing and proofreading the thesis and the many drafts that preceded it.

The financial assistance received from the National Research Council of Canada, in the form of a scholarship, is appreciated.

TABLE OF CONTENTS

CHAPTER PAGE

I INTRODUCTION 1

II AN EXAMPLE OF A MAP STORAGE AND RETRIEVAL APPLICATION 5

2.1 Introduction 5

2.2 The Sieve Process for Planning 6

2.3 The Computer Implementation 7

2.4 Planning Objectives and their Specification 15

2.5 Interactive Map Storage and Retrieval Systems 22

III DIGITAL ENCODING OF MAP DATA 28

3.1 Introduction 28

3.2 Vector-Approximation 29

3.3 Chain-Encoding 29

3.4 Skeleton-Encoding 30

3.5 Summary 34

IV AN ANALYSIS OF STORAGE REQUIREMENTS FOR ENCODED DATA 35

4.1 Introduction 35

4.2 Goodness-of-fit 38

4.3 Processing for Display 39

4.4 Discussion 41

4.5 Conclusion 45

V DATA STRUCTURES FOR GEOGRAPHIC APPLICATIONS . . 47

5.1 Introduction 47

5.2 Representation of Region Boundaries 49

5.3 A Data Structure using Chain-Encoded Data 52

5.4 The Canada Land Inventory 53

5.5 Summary 56

CHAPTER PAGE

VI A DATA BASE OF CAMPUS RESOURCES 57
6.1 Introduction 57
6.2 A Data Base for an Interactive Environment - Design Objectives 58
6.3 The Data Base 59
6.4 Implementation 72
6.5 Discussion 76
6.6 Summary 79

VII INTERACTIVE GRAPHICS PROGRAMMING 80
7.1 Introduction 80
7.2 Communication through a Graphics Terminal 82
7.3 Software for Interaction between the User and the Application Program 84
7.4 Discussion - Design Objectives for a Man-Machine Software Interface 89

VIII A TABLE-DRIVEN INTERPRETER FOR GRAPHICS APPLICATIONS 92
8.1 Introduction 92
8.2 Console Command Language Definition 94
8.3 The Principal Modules for the Interpreter 103
8.4 The Control Routine 108
8.5 An Example - The Sieve Process 108

IX IMPLEMENTATION AND DISCUSSION 113
9.1 Introduction 113
9.2 Interaction with the Interpreter 114
9.3 Discussion 115
9.4 Modified Analysis of Input 118
9.5 Examples 123
9.6 Morrison's Language Processor 125
9.7 Discussion 126
9.8 Summary 127

CHAPTER PAGE

X CONCLUSIONS 129

BIBLIOGRAPHY 132

APPENDIX A GRAPHICS MAN-MACHINE SOFTWARE INTERFACE 137

LIST OF TABLES

Table	Description	Page
2.1	Sieve Process Commands	9
2.2	Semantics of the Command Language	11
4.1	Storage Requirements	42
4.2	Storage Requirements for Buildings Map Expressed as Ratio	44
5.1	Some Signal Codes used by Freeman	54
8.1	Component Types and the Associated Attention Variables	96
8.2	Some Sieve Process Commands	110
A.1	Syntax for Lexical Entries	139

LIST OF FIGURES

Figure		Page
2.1	Display of Objective Labels	19
2.2	Display of Range Values for One Objective	21
2.3	Building Map at Small Scale	23
2.4	Selected Portions of Roads 1 and Parking Maps at Large Scale	24
2.5	As for Figure 2.4 with Corresponding Section of Building Map Overlaid	25
2.6	The General Structure of a Map Storage and Retrieval System	26
3.1	Chain-Encoding	31
3.2	Skeleton-Encoding	33
4.1	A Portion of the Contour Map	37
4.2	Skeleton-Encoding	40
5.1	A Map of Regions	50
5.2	Data Structure Required for the Map Illustrated in Figure 5.1	51
6.1	Sample Map Data	60
6.2	Block Types	62
6.3	Macro Structure of Data Base	63
6.4	Data Structure for Map Illustrated in Figure 6.1	64
6.5	Access of Data by Feature	77
6.6	Access of Data by Sector	78
7.1	State-Diagram Representation of Man-Machine Interaction	86

Figure Page

8.1 Man-Machine Interface Schematic 93

8.2 Johnson's CALD System 101

9.1 Logical Structure of a new GRID Supervisor 119

9.2 Logical Structure of Morrison's Lexical Analyzer 120

CHAPTER I

INTRODUCTION

One of the main advantages of interactive graphics is that the user may monitor the program through execution and modify, by change or addition, the data input. It is for this reason, and the fact that the CRT may be used to display maps, that an investigation to see if the techniques of interactive graphics have a contribution to make to the site-planning process was considered.

The ultimate objective in the use of interactive graphics for campus planning could be to construct a computer model of campus resources which may be readily updated to reflect any changes to the campus. Then, when the question of a new building (parking lot, utilities system, etc.) arises, one may use the techniques of interactive graphics to view portions of the campus and, given a set of requirements (such as maximum cost, capacity and so on), determine a small number of possible solutions.

Systems of such power are obviously a long way off, and will have to be approached in a series of stages. A system of sufficient power would have to:

(1) handle the map storage and manipulation required,

(2) contain an adequate model of the resources available.

As a first step, a computer implementation of the Sieve Process, a procedure used by planners in determining the site of a new building was developed in earlier work (Deecker 1970). The implementation showed that the map storage and manipulation functions required can successfully be transferred to the machine. It also revealed a number of significant problems which have been studied in greater detail and are reported in this thesis, in particular, a study of three methods of data encoding (Chapters 3 and 4), the design of a computer model of campus resources information (Chapters 5 and 6), and the description of an interpreter for console command languages which was implemented for use with the GRID and 360/67 at the University of Alberta (Chapters 7, 8, 9).

For any interactive graphics program, the retrieval of data for display must be efficient to avoid excessive processing costs. The data must be represented economically, as the processing time required is usually directly proportional to the number of data elements required. In Chapter 4, a comparative analysis of three possible data representations is presented: vector approximation, chain-encoding, and skeleton-encoding. The results have led to conclusions substantially different from those reached by Pfaltz and Rosenfeld (1967). The difference can be partially explained by the facts that the map encoded by Pfaltz and Rosenfeld was substantially different from either map considered here, and their boundary-encoding method was also different. For the three encoding methods assessed in this thesis, their merits in storage economy are affected by the magnitude of the allowable error in relation to the scale.

Another step towards the computer implementation of the complete planning process is to achieve the ability to define adequately a computer model of available resources. With such a model, we can begin to transfer to the machine some of the procedures for evaluating potential sites. In Chapter 5, important aspects of data structures which have been used to represent maps for computer applications in a non-interactive graphics environment are reviewed. In Chapter 6, a computer model which he has developed for representing campus resources is described. The data structure and associated data management routines may be used by the programmer/planner to construct a model for the particular problem at hand, and enable some evaluation procedures to be transferred to the machine.

It is generally acknowledged that support software available for interactive graphics is inadequate for convenient development of substantial applications. The considerable time required to implement the Sieve Process in an interactive graphics environment led to the study of several aspects of interactive graphics programming. In Chapter 7, two commonly used methods for interaction between a user at an "intelligent" CRT terminal and a graphics application program are reviewed. This interaction generally takes the form of commands issued by the user at the console. Thus, we term the language a Console Command Language (CCL).

The author has developed a formalism for the definition of such a CCL. A unique aspect of this formalism, described in Chapter 8, is the definition of user actions at the terminal as "terminal symbols" in the grammar defining the CCL. A table-driven interpreter utilizing this formalism has been developed for use with any interactive graphics program. As a result, the construction of the man-machine interaction component of a graphics application program is greatly simplified. Use of this software imposes a highly-desirable modularization on the applications program, and each module may be developed independently. The implementation of the interpreter is briefly described in Chapter 9, with a more detailed description in the Appendix.

CHAPTER II

AN EXAMPLE OF A MAP STORAGE AND RETRIEVAL APPLICATION

2.1 INTRODUCTION

In this chapter, a computer implementation of the Sieve Process, a procedure used by planners in determining the site of a new building, is described. While the process is applicable to a wide range of situations, the inventory, that is the data required by the planner, is completely dependent upon the type of structure to be built and the terms of reference laid down by the client. The inventory given here is that required to determine the site of the building in a university environment. A computer implementation of the Sieve Process using the computer system available at the University of Alberta, an IBM 360/67 computer and a Control Data GRID was developed earlier (Deecker 1970).

While designed as an implementation of the Sieve Process, the system is effectively a general-purpose interactive map storage and retrieval system. Its design and implementation revealed a number of problems common to map storage and retrieval applications. Two of these problems have been studied independently:

(1) The question of choosing from alternative digital encoding methods (Chapters 3, 4).

(2) The design of a structured data base (Chapters 5, 6).

In addition, the work done on the original system suggested some software developments which should be of value for interactive graphics programming generally.

The system described in this chapter has, since its description in the earlier thesis (Deecker 1970), been reconstructed using the software aids to be described in Chapters 8 and 9. The commands <list objectives>, <display values>, <modify value>, and <erase display> have also been added since the initial implementation.

2.2 THE SIEVE PROCESS FOR PLANNING

In discussing resources, we use the following terminology. A "feature" is a particular type of resource; for example, PARKING and TOPOGRAPHY are different features. A specific entity, for example a particular parking lot, is called a resource entity.

The key reference for the campus planner is a BASE MAP of the university, which outlines all existing buildings, roads and major tree lines. For each feature the planner draws a separate map, each composed of regions coloured white or black. Regions suitable for a new building with respect to the feature are white, while regions unsuitable are black. For example, on a map for buildings, the site of an existing building must be black, and a vacant lot would be white.

The planner draws these maps on transparent plastic. When the maps are overlayed, resultant dark zones indicate the used portions of the campus. The unshaded zones indicate unused portions of the campus. The planner may determine which features (generally two or three) are most important in the context of his current problem. By overlaying the maps for these features first, the main areas of interest can be located. By considering the remaining features, these areas may be refined into a list of several alternative sites. By viewing a map of land sites superimposed on the resource maps, the planner is able to narrow his investigation to one or two alternative sites. At this point a more thorough investigation may be considered to determine the best location for the building. This investigation may involve other items such as climate, noise nodes etc., for which a detailed analysis other than for a particular location would be expensive.

2.3 THE COMPUTER IMPLEMENTATION

The author has observed that, for many interactive graphics applications, there is a strong tendency for the programmer to organize his system by first specifying the functions he wants to make available, and then specifying the means by which the user at the terminal can request these functions. Thus, we see the programmer writing a set of commands and then writing a program that is effectively an interpreter of this command language. Since the language is for use at the console, we call this language a Console Command Language.

The capabilities of the system, from the user's point of view, may be understood from a systematic description of the syntax and semantics of the command language.

2.3.1 SYNTAX AND SEMANTICS OF THE COMMAND LANGUAGE

For the Sieve Process, a relatively simple command language has been developed. Table 2.1 gives a syntactic description of the commands available. The basic labels (e.g. DISPLAY) appearing in the syntax may be either words appearing on the screen or identifiers for various function keys.

For any map displayed on the screen there are two reference tags: the centre of view, and the scale of presentation. One can consider the display as a twelve inch square window showing one portion of the map. The point on the map which corresponds to the centre of the window is called the centre of view. The scale of presentation is the scale of the displayed map relative to the scale of the encoded map. Initially, the centre of view is set to the centre of the encoded maps, and the scale of presentation is set to one. With this in mind, the reader may consult Table 2.2 and see what facilities the commands offer to the user.

Table 2.1

Sieve Process Commands

<command>	→<display>|<list>|<create>|<link>|<scale>| <plot>|<erase>|<remove>|<evaluate>| <overlay>|<window>|<centre>|<delete>| <restart>|<end>|<list objectives>| <display values>|<modify value>| <erase display>
<display>	→DISPLAY<mapname>
<list>	→LIST-MAPS{AVAILABLE|ON-SCREEN}
<create>	→CREATE<mapname1>
<link>	→CATENATE<mapname><line>
<scale>	→SCALE{UP|DOWN}
<plot>	→PLOT
<erase>	→ERASE<data>
<remove>	→REMOVE<mapname>
<evaluate>	→EVALUATE<x y><objective>
<overlay>	→OVERLAY<mapname>
<window>	→WINDOW<number>
<centre>	→CENTRE<x y>
<delete>	→DELETE<mapname>
<restart>	→RESTART
<end>	→STOP
<list objectives>	→LIST OBJECTIVES
<display values>	→DISPLAY<label>
<modify value>	→MODIFY<value><newvalue>
<erase display>	→ERASE{<label>|<value>}
*<mapname>	→a lightpen pick of one of the mapnames displayed on the screen.

Table 2.1 (continued)

*<mapname1> →at most eight alphanumerics typed in from the keyboard.

*<line> →a string of vectors input with the lightpen.

*<data> →a lightpen pick of map data displayed on the screen.

*<x y> →a point indicated with the lightpen.

*<objective> →a lightpen pick of one of the objective labels displayed on the screen.

*<number> →a typed digit from 0 to 8.

*<label> →a lightpen pick of one of the objective labels displayed on the screen.

*<value> →a lightpen pick of any value displayed under the heading RANGE (see Figure 2.2).

*<newvalue> →a number typed in from the keyboard.

* The basic software (Jackson 1972) in the terminal allows the following types of input:

(1) Lightpen Pick

(2) Function Key

(3) Character String

(4) Point String

(5) Vector String.

Table 2.2

Semantics of the Command Language

COMMAND	RESULTING ACTION
DISPLAY<mapname>	Display on the screen the map titled <mapname>, where <mapname> is picked, by means of the lightpen, from a list of map names displayed on the screen. The centre of view and the scale of presentation are not changed.
LIST-MAPS{AVAILABLE|ON-SCREEN}	List on the screen the name(s) of the map(s) displayed if the word ON-SCREEN is picked, or list on the screen the name(s) of the map(s) available (that is, in storage) if the word AVAILABLE is picked.
CREATE<mapname1>	Add to the list of map names the name <mapname1>, which is entered on the alphanumeric keyboard, and create an empty file for data which may be inserted by use of the <link> command.
CATENATE<mapname><line>	Include with the data of a previously defined map <mapname> the new <line>* drawn on the screen. The user has the ability to add data to an existing file.
SCALE{UP|DOWN}	Change the scale of presentation. The centre of view remains unchanged. The scale of presentation is multiplied by a factor of two or one-half depending on whether UP or DOWN is picked with the lightpen.
PLOT	Plot on the Calcomp Plotter a hard-copy of the items currently displayed on the screen.
ERASE<data>	Erase from the map file the line which contains the <data> point picked with the lightpen. For example, if the user points to a line defining a parking lot, the outline of the parking lot will be deleted from the map file.

* Note that <line> is a string of concatenated vectors drawn with the lightpen (see Table 2.1).

Table 2.2 (continued)

REMOVE<mapname>	Remove from the screen the map <mapname> that was previously displayed using <display> or <overlay>.
EVALUATE<x y><objective>	Make an evaluation with respect to the point <x y> and the objective labelled <objective>. "Evaluation" implies the application of some function by a user-supplied subroutine. A simple example could be a function determining the distance between the designated point and the nearest utility line.
OVERLAY<mapname>	Add to the items currently displayed on the screen the map <mapname>.
WINDOW<number>	Set the centre of view to the centre of the section chosen and multiply the scale of presentation by a factor of three. (The portion of the screen in which maps are displayed is divided into nine sections. The integer designating the section the user wishes to see is typed on the keyboard.)
CENTRE<x y>	Set the centre of view to the point <x y>. The scale of presentation remains unchanged.
DELETE<mapname>	Delete from the system all information concerning the map <mapname>.
RESTART	Set the scale of presentation to one. Set the centre of view to the centre of the encoded maps. Clear the screen.
STOP	Terminate the session.
LIST OBJECTIVES	Display on the screen the labels for each of the objectives known to the system.
DISPLAY<label>	Display on the screen the range of values for the objective <label>.

Table 2.2 (continued)

MODIFY<value><newvalue>

Change any RANGE parameter from <value> to <newvalue>, the latter being a number typed in from the keyboard.

ERASE{<label>|<value>}

Erase from the screen the display of the <label>s of the objectives, or the display of the <value>s for any one objective.

2.3.2 AN EXAMPLE

The facilities available through use of the command language may be best understood from an example of how it might actually be used by the planner. The commands which may be most used in each part of a session are given in brackets at the end of that part.

2.3.2.1 SESSION ONE

(1) Check out each of the resource maps and the BASE MAP. Make any corrections deemed necessary.
(<erase>, <display>, <link>)

(2) Insert any extra maps needed for the study.
(<create>, <link>)

(3) Input some planning objectives to the system.
(<display values>, <modify value>)

(4) Determine which regions are possible building sites and draw one map containing these regions. This may be done by viewing the maps as a series of overlays and considering how well various regions fulfill the requirements.
(<overlay>, <window>, <scale>, <create>, <link>)

(5) Make a hard-copy of this map for discussion with the client. (<plot>)

2.3.2.2 SESSION TWO

(1) Erase from the map of alternatives those areas ruled out by the client. (<erase>)

(2) Evaluate the remaining alternatives to determine how well they fulfill the planning objectives. "Evaluation" may be a complex mixture of visual assessment from displayed maps, use of pre-programmed functions applied to the data base, or work done by the planner away from the machine. This important topic is discussed later in this chapter. (<evaluate>)

(3) Eliminate those alternatives which are found to be undesirable on evaluation. (<erase>)

(4) Obtain a hard-copy map showing the remaining alternatives. (<plot>)

2.3.2.3 SESSION THREE

(1) Incorporate into each resource map the changes that would result if the building were constructed at the site under consideration. (<link>, <erase>)

(2) Overlay the resource maps to check for major problems (e.g. a shortage of parking facilities) which might result. (<overlay>, <window>, <scale>)

(3) Choose the current site as the building site, or repeat from Session Two to choose another site for study.

2.4 PLANNING OBJECTIVES AND THEIR SPECIFICATION

The four new commands were added to enable the user to input some planning objectives to the system.

In the next two subsections, we discuss planning objectives and the implementation of the four new commands.

2.4.1 EXAMPLES OF PLANNING OBJECTIVES

The computer implementation of the Sieve Process was designed to answer many potential questions from the planner. For example, the question "Is this site too close to existing buildings?" may be answered by viewing an overlay of the BUILDINGS map and a map outlining the proposed site. If the site outline intersects with any section of the BUILDINGS map the proposed site is too close to the existing buildings.

Other questions, such as, "What is the area of this site?", or "What is the distance between the site and a utility tunnel?" may be answered by means of simple subroutines. One subroutine may determine the area of a site and display the answer on the screen, and another determine the distance between two points selected by the user.

Answering many questions, such as, "Is there sufficient parking nearby?", requires much more information than the current system has available. There may be parking nearby that is heavily used by patrons of existing structures. To answer the question properly, we need, as a minimum, current usage patterns of parking facilities over the campus, and an indication of the extent to which patrons of the nearby lot(s) can be accommodated elsewhere without inconvenience.

The planner is attempting to satisfy several objectives. For some objectives, "evaluation" may involve simply viewing overlays of the features concerned; others may require the computational power of the computer.

At the present time, this evaluation technique can be formalized only to a small degree. Requests for evaluation appear to follow the basic format:

"EVALUATE this <site> with respect to this <objective>".

The selection of an objective is dependent on the type of structure under consideration. For example, if a new parking structure is being considered, an important objective may be "Must be on the perimeter of the campus". If the site of a new law faculty building is to be chosen, the above objective would not apply.

One can formalize methods for input to the system of information regarding many planning objectives. However, heuristic programming techniques are used for evaluation of particular sites with respect to these objectives. To use these techniques efficiently there must be available a structured data base of campus resource information.

2.4.2 INPUT OF PLANNING OBJECTIVES

For any problem of land-site-selection, the planner realizes that objectives may not be completely fulfilled. However, there are minimum acceptance criteria for any one objective. For example, an objective may be that the land site have an area of 100,000 sq. ft., but 95,000 sq. ft. may be satisfactory, and 90,000 sq. ft. unacceptable. The overall rating of a site with respect to several objectives is most important. One common way of obtaining an overall view is to rate a site according to how well it satisfies each of a number of objectives, and to consider the average rating for several objectives.

The computer implementation of the Sieve Process allows the planner to rate, on a scale from 9 to 1, values which are output from evaluation procedures. To describe an objective to the system we need:

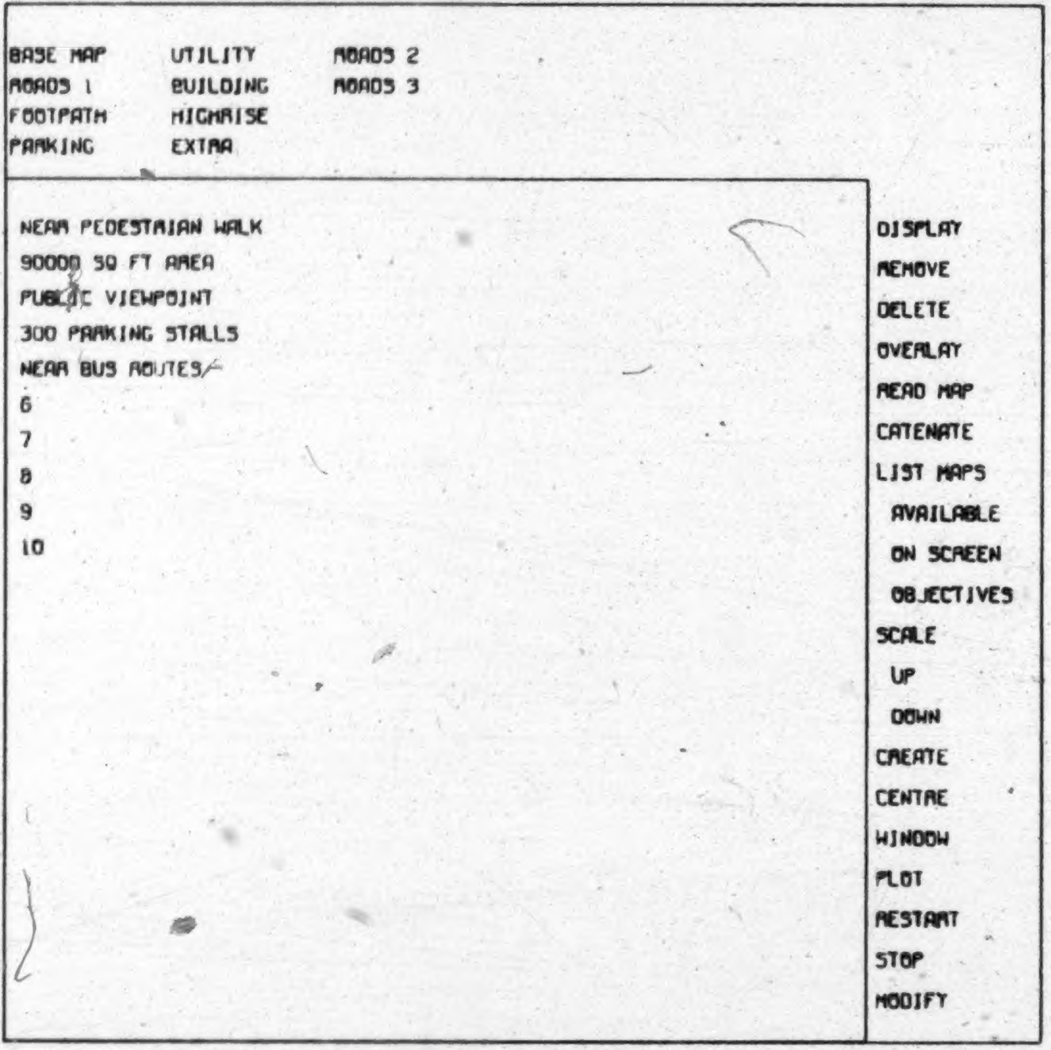
(1) a label, which is to be displayed on the CRT, (e.g. NEAR PEDESTRIAN WALK in Figure 2.1);

(2) the name of the function subprogram module to be used for evaluation;

(3) ratings on a scale from 9 to 1 for the value range of the function given in (2) above.

Suppose, for example, that the objective is that the building "Must be near major pedestrian access routes". The programmer may write a routine (perhaps called PEDES), which, when given a particular site for evaluation, determines the distance between the site and the nearest pedestrian routes. The range of values for this function may be from 0 to 200. That is, the pedestrian path may be from 0 to 200 feet away from the perimeter of the site under consideration. The planner may then rate values within this range on a scale from 9 to 1.

We require a label that is to be displayed on the CRT. By picking with the lightpen: the word "EVALUATE", a particular site, and the label "NEAR PEDESTRIAN WALK", the routine PEDES is selected for execution.



```
BASE MAP       UTILITY        ROADS 2
ROADS 1        BUILDING       ROADS 3
FOOTPATH       HIGHRISE
PARKING        EXTRA

 NEAR PEDESTRIAN WALK                                          DISPLAY
 90000 SQ FT AREA                                              REMOVE
 PUBLIC VIEWPOINT                                              DELETE
 300 PARKING STALLS                                            OVERLAY
 NEAR BUS ROUTES                                               READ MAP
 6                                                             CATENATE
 7                                                             LIST MAPS
 8                                                               AVAILABLE
 9                                                               ON SCREEN
 10                                                              OBJECTIVES
                                                               SCALE
                                                                 UP
                                                                 DOWN
                                                               CREATE
                                                               CENTRE
                                                               WINDOW
                                                               PLOT
                                                               RESTART
                                                               STOP
                                                               MODIFY
```

Figure 2.1 Display of Objective Labels

To describe this objective to the system the following data is required:

module - PEDES

label - NEAR PEDESTRIAN WALK

Range	Point Score
0-20	9
20-30	8
30-35	7
35-38	6
38-45	5
45-60	4
60-90	3
90-120	2
120-200	1

Only the range values may be modified from the CRT. Since the programmer must supply a routine for each objective, the user does not have the ability to change the objectives dynamically. Rather, the objectives and evaluation routines are supplied to the system for each session.

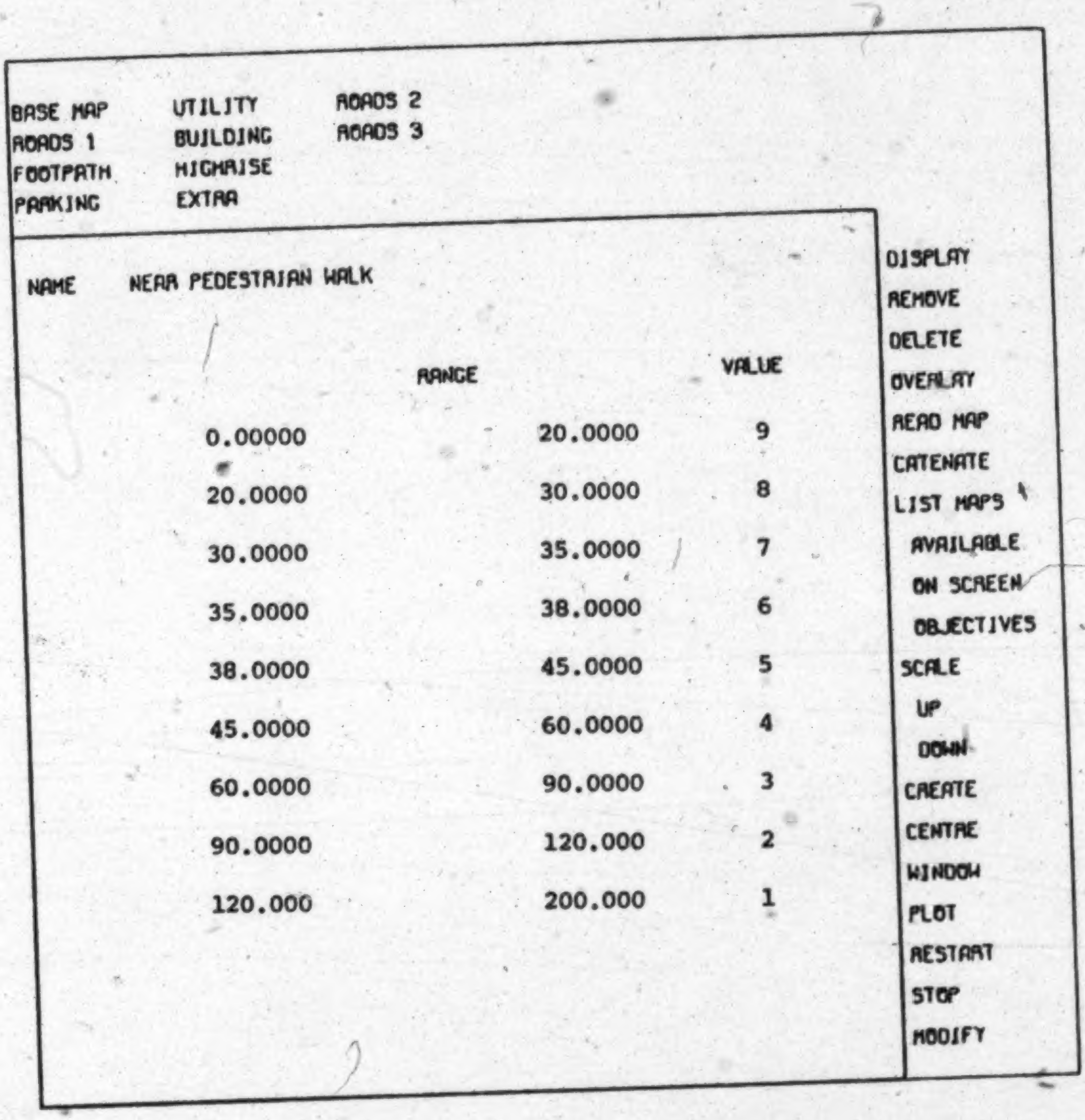
Figure 2.1 shows the result of using the command:

```
LIST OBJECTIVES
```

The labels of all currently defined objectives are displayed on the screen. Use of the command:

```
DISPLAY <label>
```

gives the user a display of the scale of values for a particular objective (Figure 2.2). Any values under the heading RANGE may be modified by means of the MODIFY command. The display of the list of objective labels or of the values for a particular objective may be erased from the screen by


BASE MAP
ROADS 1
FOOTPATH
PARKING
UTILITY
BUILDING
HIGHRISE
EXTRA
ROADS 2
ROADS 3
DISPLAY
REMOVE
DELETE
OVERLAY
READ MAP
CATENATE
LIST MAPS
AVAILABLE
ON SCREEN
OBJECTIVES
SCALE
UP
DOWN
CREATE
CENTRE
WINDOW
PLOT
RESTART
STOP
MODIFY

```
NAME     NEAR PEDESTRIAN WALK

                    RANGE                       VALUE

          0.00000             20.0000             9
          20.0000             30.0000             8
          30.0000             35.0000             7
          35.0000             38.0000             6
          38.0000             45.0000             5
          45.0000             60.0000             4
          60.0000             90.0000             3
          90.0000             120.000             2
          120.000             200.000             1
```

Figure 2.2 Display of Range Values for one Objective

use of the ERASE command.

2.5 INTERACTIVE MAP STORAGE AND RETRIEVAL SYSTEMS

The system described allows:

(a) Stored maps, or sections of these maps, to be viewed on the CRT, in any combinations, at varied scales, and centred about specific points.

(b) Stored maps to be modified on-line.

(c) Pre-programmed functions (for example, area evaluation) to be applied to maps, or parts of maps, as specified by the operator.

(d) Hard-copy plots to be produced on request.

It can be seen that these are functions of a reasonably general map storage and retrieval system. For example, after picking with the lightpen the words DISPLAY and BUILDING, the operator is shown the map of buildings (Figure 2.3). Maps in storage are visualized as being drawn on a large, imaginary surface, with the CRT used as a "zoom lens" enabling an operator to view any map segment at any desired scale. Parker (1966) gives an illustration of this technique. Figure 2.4 shows parts of a map of roadways and parking; Figure 2.5 shows the same area with part of the buildings map overlaid.

Many of the problems encountered in developing this system are common to a wide range of map storage and retrieval applications. A generalized schematic for a map storage and retrieval system is shown in Figure 2.6. For this type of system, a data base contains map data (image data) and descriptive data, accessible in either batch or interactive

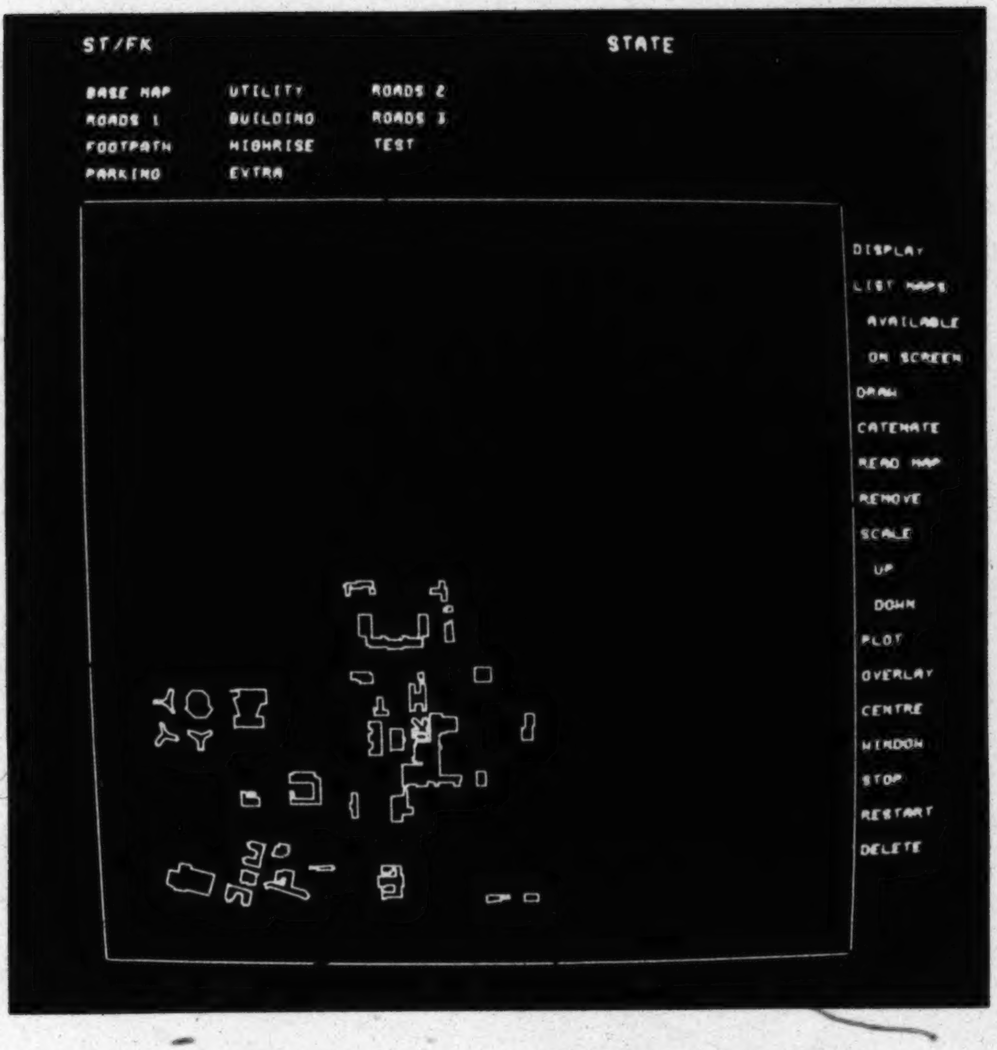

ST/FK
STATE
BASE MAP
UTILITY
ROADS 2
ROADS 1
BUILDING
ROADS 3
FOOTPATH
HIGHRISE
TEST
PARKING
EXTRA
DISPLAY
LIST MAPS
AVAILABLE
ON SCREEN
DRAW
CATENATE
READ MAP
REMOVE
SCALE
UP
DOWN
PLOT
OVERLAY
CENTRE
WINDOW
STOP
RESTART
DELETE

Figure 2.3 Building Map at Small Scale

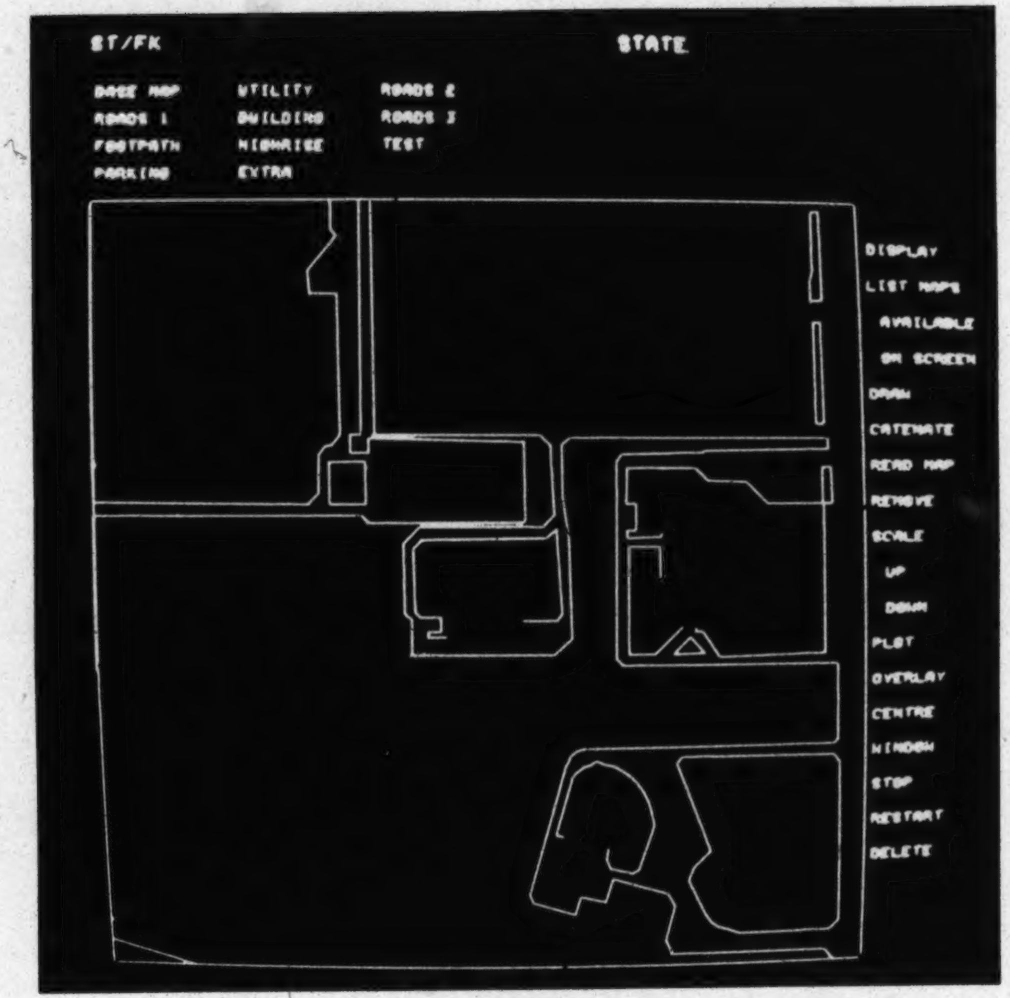

ST/FK
STATE
BASE MAP
ROADS 1
FOOTPATH
PARKING
UTILITY
BUILDING
HIGHRISE
EXTRA
ROADS 2
ROADS 3
TEST
DISPLAY
LIST MAPS
AVAILABLE
ON SCREEN
DRAW
CATENATE
READ MAP
REMOVE
SCALE
UP
DOWN
PLOT
OVERLAY
CENTRE
WINDOW
STOP
RESTART
DELETE

Figure 2.4 Selected Portions of Roads 1 and Parking Maps at Large Scale

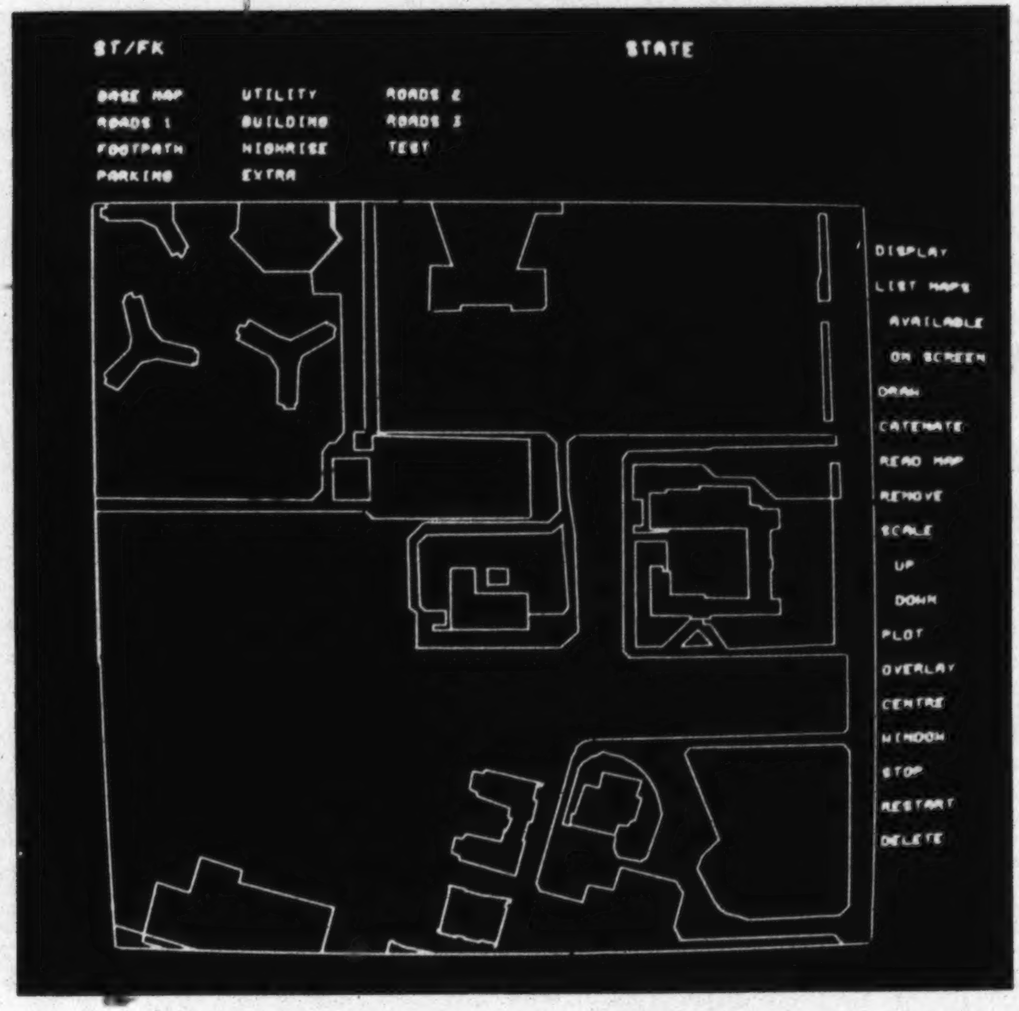

ST/FK
STATE
BASE MAP
ROADS 1
FOOTPATH
PARKING
UTILITY
BUILDING
HIGHRISE
EXTRA
ROADS 2
ROADS 3
TEST
DISPLAY
LIST MAPS
AVAILABLE
ON SCREEN
DRAW
CATENATE
READ MAP
REMOVE
SCALE
UP
DOWN
PLOT
OVERLAY
CENTRE
WINDOW
STOP
RESTART
DELETE

Figure 2.5 As for Figure 2.4, with Corresponding Section of Building Map Overlaid

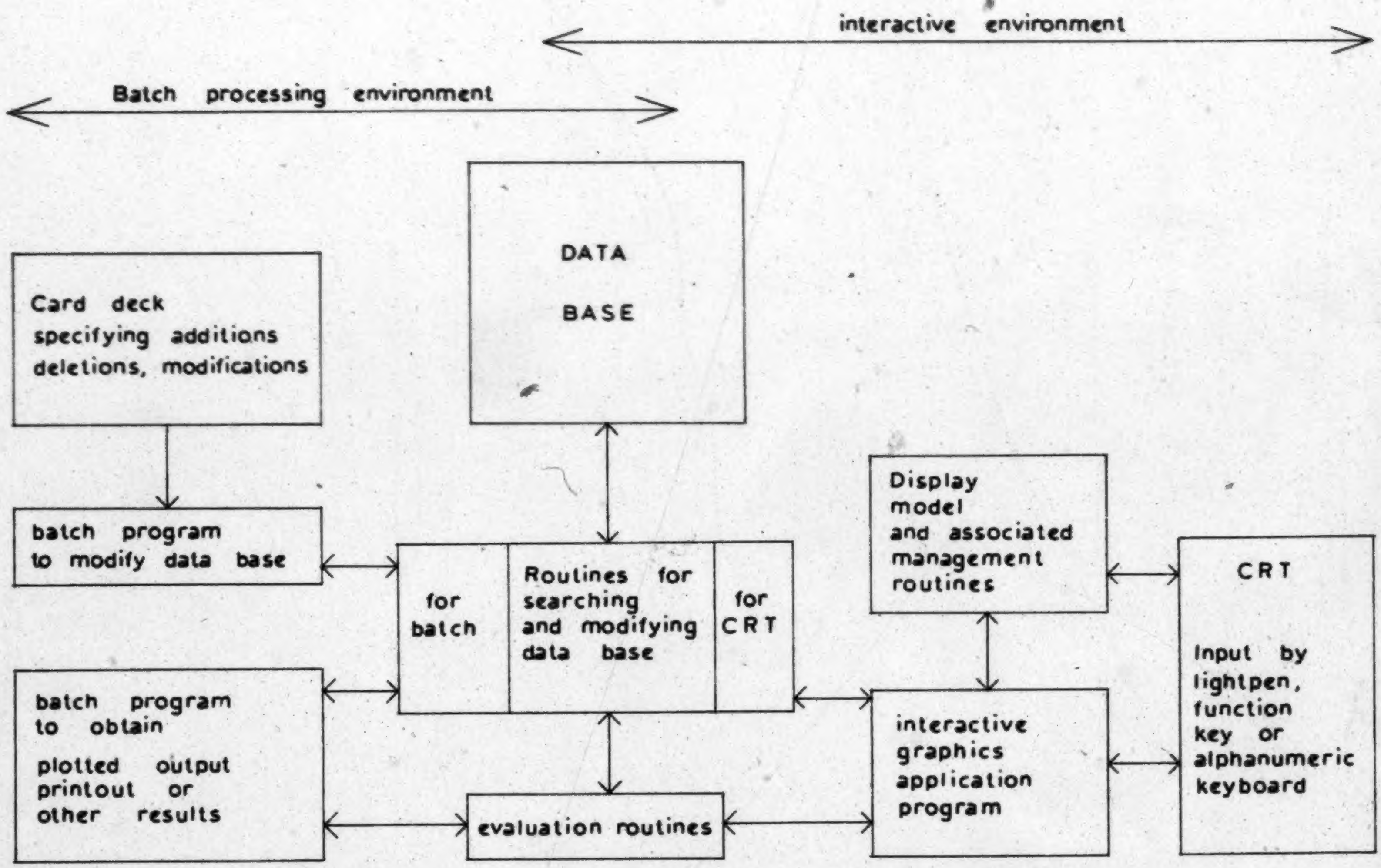

interactive environment
Batch processing environment
DATA BASE
Card deck specifying additions deletions, modifications
batch program to modify data base
for batch
Routines for searching and modifying data base
for CRT
Display model and associated management routines
CRT
Input by lightpen, function key or alphanumeric keyboard
batch program to obtain plotted output printout or other results
evaluation routines
interactive graphics application program

Figure 2.6 The General Structure of a Map Storage and Retrieval System

mode. In batch mode, card decks containing information for updating the data base (or requests of plotted output or listing of data) are processed by the computer. In interactive mode, the user sitting at the CRT, by means of a CCL, may request display of any information from the data base. He may also, by means of a lightpen, function keys, alphanumeric keys or other devices, update the information in the data base. Evaluation routines are available to the user. He may, by using the CCL, request that an evaluation procedure be applied to data indicated with the lightpen and have the result displayed on the CRT.

If the planner uses the map-viewing functions as freely as he would use hard-copy plans, then the form in which data is represented is of fundamental importance in the interactive environment. Certain operations such as "windowing", scaling and display file generation will be done very frequently. The time required to apply these operations must be minimized, if an economic implementation is to be achieved. "Data representation", that is the design of the data base, involves two components: digital encoding (to be discussed in Chapters 3 and 4) and data structuring (to be discussed in Chapters 5 and 6).

CHAPTER III

DIGITAL ENCODING OF MAP DATA

3.1 INTRODUCTION

In the next two chapters, the results of an assessment of a number of methods of digital representation of map data are reported. Efficient encoding is vital for an interactive storage and retrieval system, because each of disc storage, core storage and disc-core transfer time is likely to be a scarce resource. Further, in an interactive retrieval system, certain operations on stored data, such as "windowing", scaling and display file generation, must be done often.

For example, each time the user of the Sieve Process uses any one of the commands <scale>, <window>, <centre>, <display> or <overlay> one or more map files must be processed in the following manner. The data is retrieved, "windowed", scaled and a display file created for that portion which is to appear on the screen. The time to apply these operations, which are very different from those required in a non-interactive system, is usually proportional to the number of data elements representing each map.

To choose, from a number of possibilities, a method of representation suitable for interactive work the following questions must be asked:

(1) What storage is required for the same data in each form?

(2) How quickly can the frequently used operations, including display file generation, be applied to

data represented in each form?

The relative ease with which data can be encoded initially may be neglected, since initial encoding is a once-only operation.

The author has examined three methods of data representation: vector-approximation, chain-encoding, skeleton encoding. In this chapter a description of each of these three methods is presented. In Chapter 4 the results of encoding two different types of map data in each of the three forms of representation are examined.

3.2 VECTOR-APPROXIMATION

A curve may be approximated by variable-length straight line segments. The data that represents the curve is a sequence of coordinates, absolute or relative, of points on the curve. Mezei used this format for his SPARTA package (Mezei, 1968).

Original map data may be encoded by use of a digitizer. Usually the operator of the machine manually selects the end points of the vectors.

3.3 CHAIN-ENCODING

An arbitrary curve may be represented by a set of vectors restricted in both size and direction.

Freeman (1961b) described the process of encoding a line as follows. "Consider a uniform rectangular grid superimposed on an arbitrary curve..... At each point of intersection between the curve and a line of the rectangular grid, the distances to the two grid nodes adjacent to this intersection are determined. The closer of the two grid nodes is selected as a curve point. (A curve point is defined as a grid node used in the representation of the curve.)..... As the curve is traversed from end to end, if the coordinates of one curve point (m,n) are known, the next curve point (i,j) can assume only one of eight possible neighbouring positions on the grid which satisfy the relations:

$$|i-m| \leq 1$$

$$|j-n| \leq 1$$

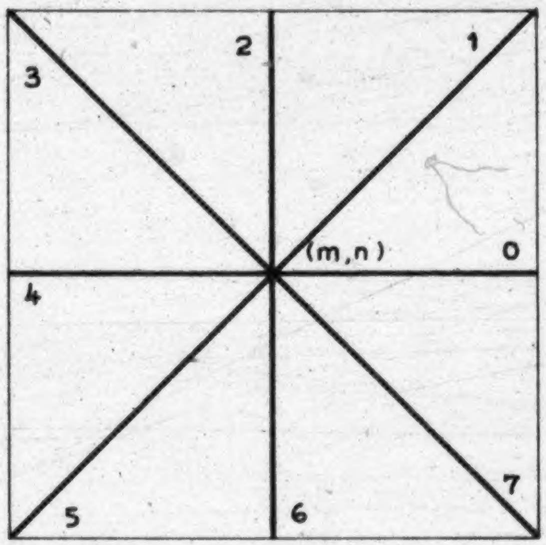
The eight grid-point positions surrounding a given point (m,n) may be labelled from 0 to 7....." The grid is illustrated in Figure 3.1a. Thus the complete curve can be described on the grid system by giving the coordinates of the starting point, the grid size, and the code for each subsequent curve point (see Figure 3.1b).

3.4 SKELETON-ENCODING

Skeleton-encoding varies from the previous two forms of data representation in that for any arbitrary region the encoded data represents the area enclosed by the region rather than the outline of the region.

The procedure for encoding any region is given in Rosenfeld and Pfaltz (1966). Assume a matrix of points with Cartesian coordinates (i,j) to be overlaid on the region. A rectangular matrix of elements ($a_{i,j}$) is set up where:

$a_{i,j}=1$ if the point with coordinates (i,j) is inside the region, and 0 otherwise.



2
1
3
(m,n)
0
4
7
5
6

Figure 3.1a Chain-Encoding Grid

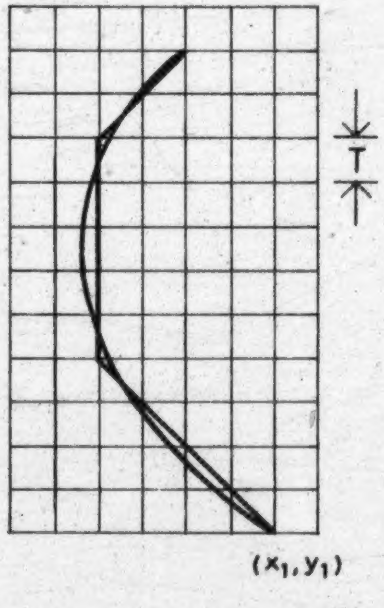


Figure 3.1b The Curve is Encoded as x_1,y_1,T,3333222211

If $P_1 = a_{i_1,j_1}$ and $P_2 = a_{i_2,j_2}$ are two elements of the matrix, the "distance" from P_1 to P_2 is defined as

$$d(P_1,P_2) = |i_1-i_2| + |j_1-j_2|.$$

Given a digitized picture whose elements are 0 or 1, we may then construct a distance transform of the picture in which each element has an integer value equal to its distance from the set of 0's. Let

$$\begin{aligned} f_1(a_{i,j}) &= 0 \quad \text{if } a_{i,j}=0, \\ &= \min(a_{i-1,j}+1,\ a_{i,j-1}+1) \quad \text{if } (i,j)\neq(1,1) \text{ and } a_{i,j}=1, \\ &= m+n \quad \text{if } (i,j)=(1,1) \text{ and } a_{1,1}=1, \end{aligned}$$

$$f_2(a_{i,j}) = \min(a_{i,j},\ a_{i+1,j}+1,\ a_{i,j+1}+1).$$

By applying, to the encoded matrix $a_{i,j}$ representing the region, f_1 in forward raster sequence (that is, in the sequence $a_{1,1}\ a_{1,2}$ ..... $a_{1,n}\ a_{2,1}$ ..... $a_{2,n}$ .....$a_{m,1}$ ..... $a_{m,n}$) then applying f_2 in reverse raster sequence (that is, in the sequence $a_{m,n}\ a_{m,n-1}$ ..... $a_{m,1}\ a_{m-1,n}$ ..... $a_{m-1,1}$ ..... $a_{1,n}$ ..... $a_{1,1}$) the modified matrix is the distance transform of $a_{i,j}$.

The function

$$\begin{aligned} f_3(a_{i,j}) &= a_{i,j} \quad \text{if none of } a_{i-1,j},\ a_{i+1,j},\ a_{i,j-1},\ a_{i,j+1} = a_{i,j}+1, \\ &= 0 \quad \text{otherwise,} \end{aligned}$$

may then be applied to the distance transform to obtain the set of local maxima of the distance transform. This set is called the skeleton of the region so encoded.

Figure 3.2 shows the set of local maxima or skeleton of a sample region.

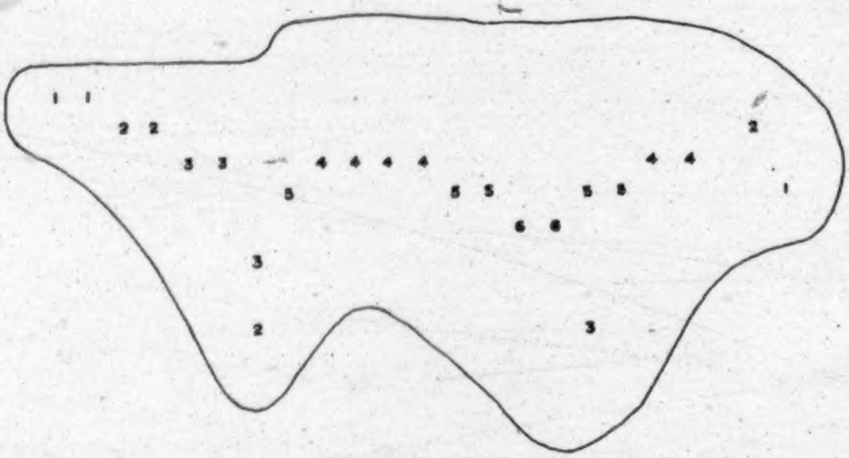

1 1
2 2
3 3
4 4 4 4
5
5 5
6 6
5 5
4 4
2
1
3
2
3

Figure 3.2 A Sample Region and the set of Local Maxima

Each element $a_{i,j}$ in the set of local maxima may be stored in memory as a triplet (i,j,r) where

i,j are the coordinates,

r is the value of $a_{i,j}$.

3.5 SUMMARY

We have described three substantially different digital encoding methods for map data. In Chapter 4, we examine the use of these three representations for two different types of map data, with particular reference to the requirements for store space.

CHAPTER IV

AN ANALYSIS OF STORAGE REQUIREMENTS FOR ENCODED DATA

4.1 INTRODUCTION

The study of various forms of digital representation of map data was initiated in an attempt to answer the question "Given a large and arbitrary collection of map data to be used in an interactive system, and a number of possible encoding methods, what is the 'best' encoding method for that data?"

In an initial study of various frequently-used functions of data manipulation (Deecker 1970) it was seen that while some functions were much simpler to apply to data represented in one form than another, the processing time required to apply any function is likely to vary linearly with the number of data elements involved. For example, to determine the area of a region represented by chain-encoded data involves one addition operation for each element in the chain, whereas for data in a skeleton form, we need to sum the areas of all maximal neighbourhoods, while subtracting overlap with adjacent neighbourhoods. However, if there are significantly fewer skeleton points than chain-encoded vectors, the total processing time required could be less.

In addition, a number of cost factors, such as those for core, disc, and disc-core transfer time are directly related to the number of data elements involved. The author therefore concluded that a key question to study is: "For any given sample of data, how many elements of data are needed to represent it by different encoding methods?"

An evaluation was made for two maps of different types, at three levels of accuracy of approximation. One, a map of buildings, is part of the map shown in Figure 2.3; the other a contour map, is illustrated in Figure 4.1.

For a valid comparison of different encoding methods, one must ensure that each representation approximates the original data "equally well". Two problems must be considered. First, what we store in a digital computer as "original" data is already an approximation. Second, a quantitative criterion is needed for goodness-of-fit.

Each map was specified to a high degree of apparent accuracy (±0.005") - much higher than justified by the digitizing equipment used for the buildings map - and we declared these data to be the originals for purposes of comparison. To have, for an "original" contour map, data to this degree of "accuracy", a computer-generated map provided by a contour-fitting package was used.

In Section 4.2 we present a definition of goodness-of-fit for our purposes, and show how an approximation is achieved for each data representation. In Section 4.3 a discussion of one of the most frequently applied functions, that of processing for display, is presented. The reader may also consult Peucker (1972) for a discussion of the functions of data manipulation most generally used in the related area of computer cartography applications. In Section 4.4 the results of the analysis of storage requirements are discussed.

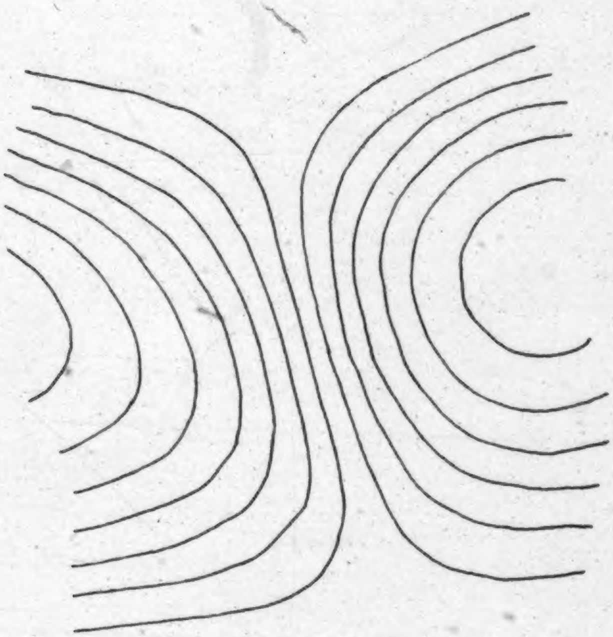


Figure 4.1 Small Fragment of the Contour Map used in the Comparison

4.2 GOODNESS-OF-FIT

We define an approximation with maximum error ϵ to be such that:

(1) each point on the original outline is no further than ϵ units from at least one point on the approximation;

(2) each point on the approximation is no further than ϵ units from at least one point on the original.

For vector-approximation, the algorithm used does not ensure that a minimum number of vectors are used. (Examination of the question of obtaining an optimum fit suggests that the iterative procedure required would take a great deal of time, which is unlikely to be justified by the improvement gained over the relatively simple procedure outlined below). Rather, it merely omits points that are not required to keep the error within the bound ϵ. The algorithm is as follows:

Assume the boundary is represented by n data points where $n>2$. Assume the data points are labelled $P_1, P_2, \ldots\ldots P_n$, and the new set of points is $P'_1, P'_2, \ldots\ldots P'_{i'}$.

```
(1)  Set $i=1$, set $i'=1$, set $P'_1=P_1$.
(2)  Set $j=3$.
(3)  Determine the coefficients of the line L joining $P_i$ and $P_j$.
(4)  If any point $P_k$, $i<k<j$, is not within $\epsilon$ of the line L go to step 5. Set $j=j+1$, if $j \le n$ go to step 3.
(5)  Set $i'=i'+1$ and $P'_{i'}=P_{j-1}$.
(6)  Set $j=j-1$. If $j=n$ halt.
```

(7) If $j<n-1$, set $i=j$ and $j=j+2$ and go to step 3.

(8) Set $i'=i'+1$ and $P'_{i'}=P_n$ then halt.

For chain-encoding, we noted in Section 3.3 that a grid was laid over the line to be encoded. To have a chain-encoded representation of a line with maximum error ϵ we need only to ensure that the node points are at most ϵ distance from the intersection of the line to be encoded with the grid. This is possible by defining the grid size T to be 2ϵ.

For skeleton-encoding, we noted in Section 3.4 that a grid was overlaid on the line to be encoded and that if the point (i,j) was within the region to be encoded the square represented by the grid-point (i,j) was deemed to be inside the region. Where a region is skeleton-encoded, the outline must be reconstructed for display. The distance ϵ is then the distance between the original outline and the reconstructed outline (Figure 4.2a). If we choose grid size of $\sqrt{2}\epsilon$, then we can meet the goodness-of-fit criterion. Figure 4.2b shows $a_{i,j}$ within the region, $a_{i,j+1}$ outside it.

4.3 PROCESSING FOR DISPLAY

The operation to be applied most frequently is the transformation of data into the form required by the display hardware. Unfortunately, the nature of the operation is both data and hardware-dependent, and generally-applicable conclusions are hard to draw. For a line drawing CRT, skeleton-encoded data must be converted to outline form for display. Generation of display files is straightforward for both vector-approximated and chain-encoded data; it is likely

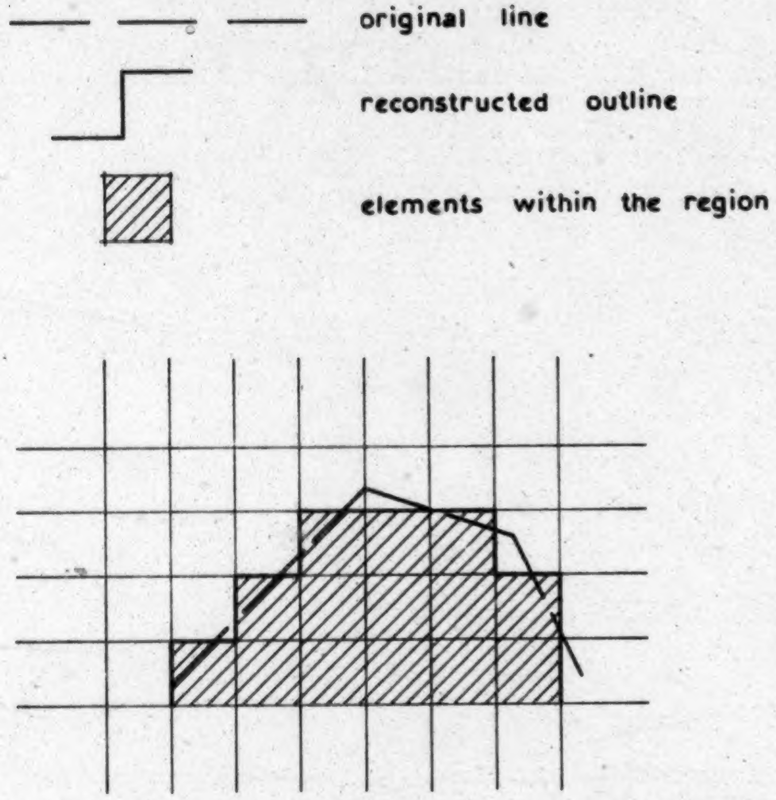

original line
reconstructed outline
elements within the region

Figure 4.2a Original and Reconstructed Outlines

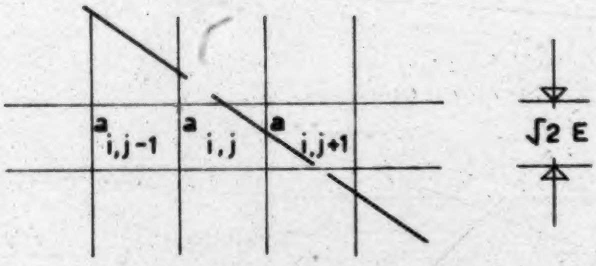

$a_{i,j-1}$
$a_{i,j}$
$a_{i,j+1}$
$\sqrt{2}\,E$

Figure 4.2b Grid Overlaid on Region Outline

to be faster for vector-approximated data because of the smaller number of data elements, but the position could be reversed if the display has an incremental line-drawing mode.

4.4 DISCUSSION

An assessment of methods of digital encoding is important for two reasons, an obvious one being storage economy. In addition there is the fact that a given processing operation must usually be applied to each element of data in turn, so that processing efficiency varies with the number of elements needed to encode the data to the required degree of accuracy.

Table 4.1 shows that for the same data encoded in each of the data representations, storage requirements can vary widely. For example, the map of buildings encoded with an ϵ of .025 requires only 93 words of storage for chain-encoded data, but over seven times as much storage (660 words) if stored as skeleton-encoded data.

This result, an admittedly extreme case, is very different from that reported by Pfaltz and Rosenfeld (1967). For an outline map of South East Asia, Pfaltz and Rosenfeld found that, "except for China and Burma, which have long straight lines as major parts of their boundaries, there are always somewhat fewer skeleton points than boundary segments". For example, the outline of Cambodia required 178 boundary segments or 135 skeleton points. However, specifying a boundary segment requires fewer bits than specifying a skeleton point, and they concluded that "the skeleton method of encoding requires somewhat more storage than do the

Table 4.1

Storage Requirements

				Vector-approximation		Chain-encoding		Skeleton-encoding	
Map	"Original"	ϵ	s[4]	Elements	/360 words[1]	Elements	/360 words[2]	Elements	/360 words[3]
Buildings	9 buildings	0.1	2.9	74	74	182	31	166	166
	156 data points	0.05	5.8	97	97	381	52	346	346
		0.025*	10.6	127	127	809	93	660	660
Contours	28 contours	0.2*		174	174	922	133	406	406
	10487 data points	0.1		211	211	1833	222	632	632
		0.05		274	274	3677	409	1085	1085

*Error values considered acceptable for each map type.

[1] Assuming that $|x_i.10^d|,|y_i.10^d| \le 2^{15}-1$ (d = number of decimal places).

[2] Assuming 10 increments per word, plus initial data for each contour.

[3] Assuming that $|x_i.10^d|,|y_i.10^d| \le 2^{12}-1$, and $r \le 2^6-1$.

[4] s = mean line length ÷ ϵ.

boundary methods".

The map encoded by Pfaltz and Rosenfeld was substantially different from either map discussed here. Their boundary-encoding method was a variant of chain-encoding in that each line segment was of variable length with slopes only "in the eight principal directions (horizontal, vertical, diagonal)". For the maps presented here, skeleton-encoding was found to be significantly less economical than either boundary-encoding method. In private discussions with the author in 1971, Pfaltz agreed with this conclusion.

Table 4.2 summarizes, for each of our maps, the relative storage requirements using each of the three methods. The Table shows that very different results were obtained for different error values. However the choice of allowable error above is not the determining factor, for the same effect would be seen if the error were kept constant and the scale of the map varied. (The choice of "scale" for encoding a map is a very arbitrary one.) We have concluded that the comparison of methods is significantly influenced by the value of the error allowed in relation to the scale. For the buildings map, representative of maps composed of long, straight lines, Table 4.1 shows values of the dimensionless quantity

$$s = \text{mean line length} \div \epsilon .$$

For maps of buildings, one wants a high degree of detail (implying a small ϵ), and our results suggest that in this case vector-approximation and chain-encoding are comparable. Chain-encoding is more attractive if the scale is decreased

Table 4.2

Storage Requirements for Buildings Map Expressed as Ratio

ϵ	s	Vector-approximation	Chain-encoding	Skeleton-encoding
.1	2.9	2.4	1	5.4
.05	5.8	1.9	1	6.7
.025	10.6	1.4	1	7.1

(or the allowable error increased). For the contour map, vector-approximation and chain-encoding are again comparable. Here, one is prepared to accept greater loss of detail (larger ϵ); chain-encoding and skeleton-encoding become relatively more attractive as the allowable error increases.

4.5 CONCLUSION

For interactive map retrieval to be economic, data must be encoded efficiently, both to achieve storage economy and to reduce processing time for the frequently applied operations. The encoding method chosen should be both efficient and efficiently exploited - that is, with a constant error.

For the three encoding methods assessed, their relative merits for storage economy are affected by the magnitude of the allowable error in relation to the scale. For the maps considered and at the accuracies desired, chain-encoding requires a little less storage than vector-approximation, which requires substantially less than skeleton-encoding.

The data representation must allow the retrieval operations, including display file generation to be done quickly. The author concluded that either vector-approximation or chain-encoding is a satisfactory choice for a general-purpose interactive retrieval system. In the system described in this thesis the method used has been vector-approximation.

There remains much work to be done. The area of data-encoding is worthy of a thesis project in its own right and we have merely shown that the issues are not as clear cut as had been hoped.

In the next chapter computer models for three geographic applications, using either chain-encoded data or data represented by vector-approximation, are reviewed.

CHAPTER V

DATA STRUCTURES FOR GEOGRAPHIC APPLICATIONS

5.1 INTRODUCTION

Hitherto, most planning has been done without extensive use of the computer. Where the computer is used, it is primarily for solution of isolated problems requiring substantial computation. For example, the computer was used by planning consultants (University of Alberta 1968) to determine an optimum column span width for a set of academic buildings.

In this case, Faculties submitted a list of required room sizes, and the maximum number of rooms of each size which would be required in any column-free area. A computer was then used to derive a graph of column span width against the number of alternative configurations possible. From this information, and costs for various span widths, a span width was chosen that allowed a maximum of flexibility in space arrangement at a reasonable cost.

Though an experimental system, the computer implementation of the Sieve Process has shown that most of the map storage and handling required for planning can be transferred to the machine. Potentially more important, it suggests that a much larger part of the process of planning could be served by a single man-machine system, provided that the problems of developing an adequate campus model and supplying algorithms to operate on the model can be solved.

The man-machine combination, with each doing what the man (at the moment) or the machine does best is a long way from the "ultimate objective" suggested at the beginning of this thesis. It would represent, however, a significant advance over a situation in which the machine is employed simply as an occasional tool.

A key problem, then, is to define a data base which contains information regarding campus resources and other data required for evaluation purposes. With this data base, we can begin to transfer to the computer more of the evaluation processes needed for solution of planning problems.

In Chapter 6 we shall describe a computer model of campus resources, including a data structure and the necessary support routines, for use in an interactive environment. However, before presenting this model, the remainder of the present chapter is devoted to a review of some aspects of a number of articles which illustrate very different ways in which map data, or more generally line data, has been handled by other researchers in an non-interactive environment. These describe:

(1) A land inventory scheme (Cook 1967) where data in the form of maps is encoded and stored.

(2) A system (Freeman 1970) by which chain-encoded data may be linked with other descriptive data.

(3) The Canada Land Inventory (CLI) project (Tomlinson 1967), a mapping of Canada by land survey and aerial photography.

5.2 REPRESENTATION OF REGION BOUNDARIES

Cook has stated that his aim in design of a data structure was to "exhibit the pictorial information contained in a map of region boundaries". This aim has been achieved by "building a structure having relations between its elements analogous to the pictorial relations between map elements".

A "region" is a connected area bounded by one or more closed curves. Region boundaries are represented by polygons. A "point" is a vertex belonging to at most two regions. A "junction" is a vertex belonging to three or more regions. In order to traverse the boundary of a region, a way of choosing the correct path beyond a junction is needed. Cook solved the problem by defining a block type "angle". For each junction there is a ring of N angles where N is the number of regions containing the junction as a boundary point. Each angle block, besides containing pointers to the adjacent elements in the two-way ring of angles, points to its region block and to the next point block or junction block on the boundary of that region.

Thus, the data structure has four distinct block types: region - which contains a pointer to one of the junctions in its boundary outline; angle; junction; and point. Only the last two block types contain coordinate data. An outer region is defined to include all the area within the frame not included in other regions. Figures 5.1 and 5.2 show a map of regions and the data structure required to encode the map.

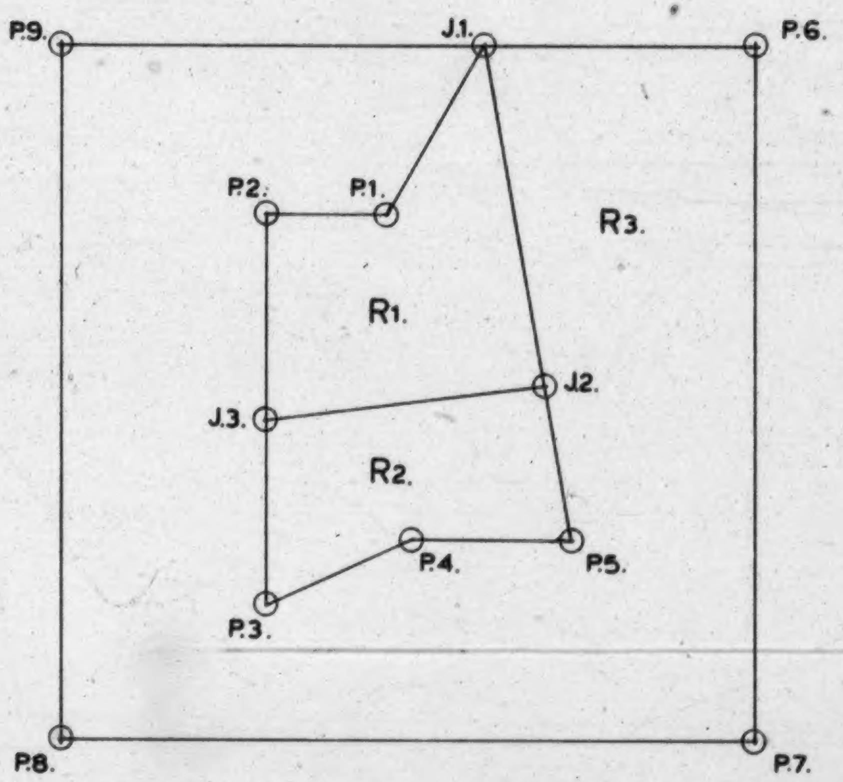
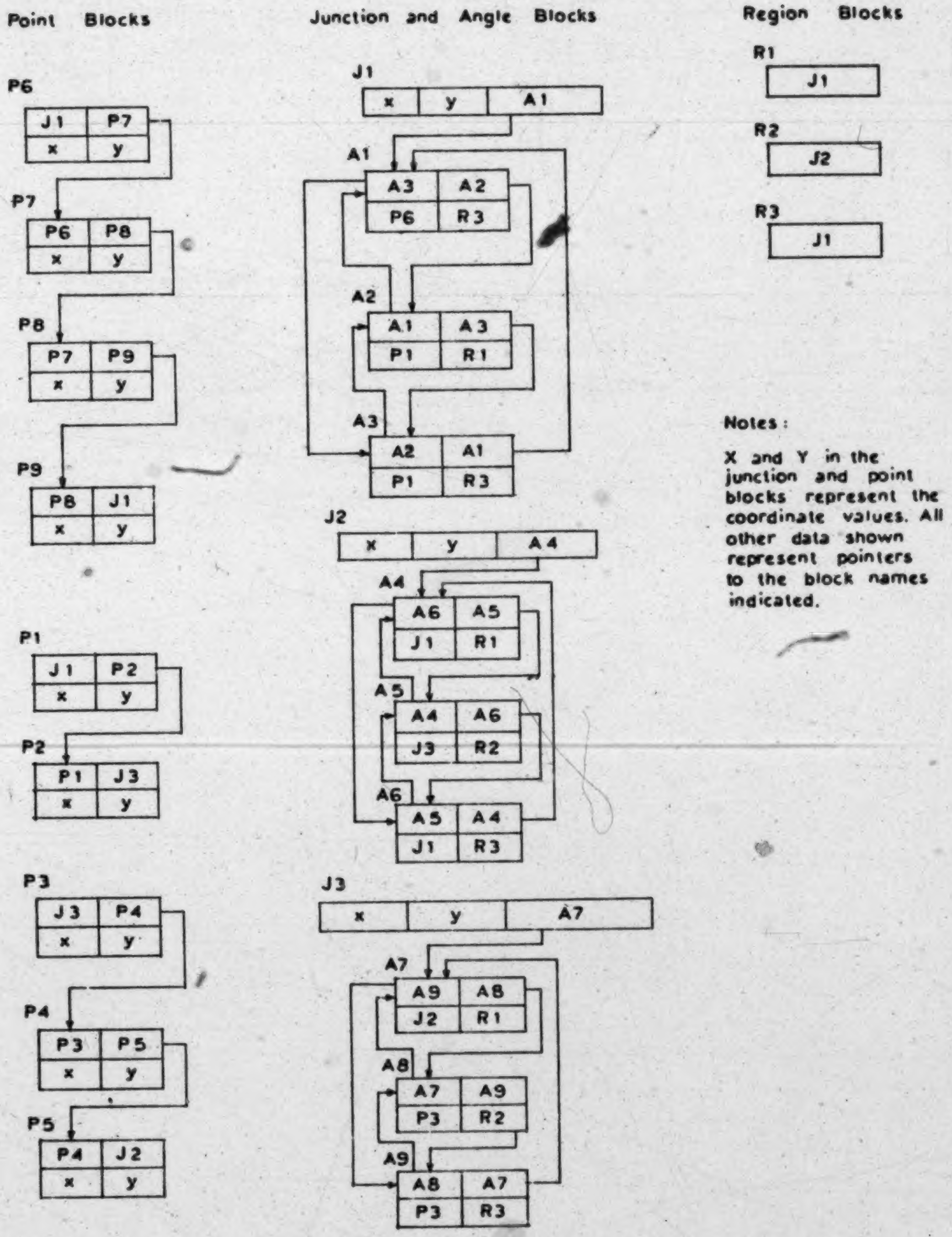

P.9.
J.1.
P.6.
P.2.
P.1.
R3.
R1.
J.2.
J.3.
R2.
P.4.
P.5.
P.3.
P.8.
P.7.

Figure 5.1 A Map of Regions


Point Blocks
Junction and Angle Blocks
Region Blocks
P6
J1 P7
x y
P7
P6 P8
x y
P8
P7 P9
x y
P9
P8 J1
x y
P1
J1 P2
x y
P2
P1 J3
x y
P3
J3 P4
x y
P4
P3 P5
x y
P5
P4 J2
x y
J1
x y A1
A1
A3 A2
P6 R3
A2
A1 A3
P1 R1
A3
A2 A1
P1 R3
J2
x y A4
A4
A6 A5
J1 R1
A5
A4 A6
J3 R2
A6
A5 A4
J1 R3
J3
x y A7
A7
A9 A8
J2 R1
A8
A7 A9
P3 R2
A9
A8 A7
P3 R3
R1
J1
R2
J2
R3
J1

Notes:

X and Y in the junction and point blocks represent the coordinate values. All other data shown represent pointers to the block names indicated.

Figure 5.2 Data Structure Required for the Map Illustrated in Figure 5.1

Access to the structure is by means of the address of any node in the structure. In general, access would be made through a table of region blocks (R_1, R_2, and R_3 in Figure 5.2).

The main advantage of this structure is that it explicitly illustrates the pictorial relations between map elements by relations between structured elements. Thus the structure appears "natural" to the user and is relatively easy to understand. However, Cook admits that its "naturalness" is achieved at the cost of excessive use of storage.

A severe drawback to use of this type of structure in an interactive environment would be the amount of processing that must be performed to obtain the outline of a region for display purposes. The operation would be both slow and, since the display function will be performed frequently, costly compared to other representations.

5.3 A DATA STRUCTURE USING CHAIN-ENCODED DATA

Freeman (1970) has suggested that map data be organized as chain-encoded data (described in Section 3.3) with all labelling, scaling and other associated information included within the chain. This may be accomplished by use of "signal" codes identified by the sequence 04. By considering Figure 3.1a, we see that the sequence 04 represents a pair of increments which double back on each other; the sequence will therefore not occur frequently in the chain of increments describing a line. In the event that the pair 04 is required, the sequence 0404 is inserted in the data. Signals 40, 15, 37, 62, could alternatively have been chosen. A few of the special codes proposed by Freeman are given in Table 5.1.

From Table 5.1, we see that one can include in a block of chain data any information that the user considers relevant. Special values may indicate nodes. Structures may therefore be developed, but in a much less natural way than is possible with Cook's approach. Contour lines may have elevation data contained within the line description.

By including descriptive data within the chain-encoded data, Freeman has produced a relatively simple technique that is economical in use of storage. At the same time, the structure may be processed in a straightforward manner to obtain a fully documented copy of the encoded map drawing.

A disadvantage of the structure is that by including the descriptive data, (e.g. grey scale value) within the image data, it takes relatively more processing if one wants to retrieve only the descriptive data. A second disadvantage is that a separate "signal code" is normally required for each type of descriptive data. For example, the signal code "0407" indicates that a serial number follows, and signal code "0424" that a grey scale value follows. Alternatively, data between the 0411 and 04127777 signals may be structured in some way, but an increase in complexity results.

5.4 THE CANADA LAND INVENTORY

The Canada Land Inventory (Tomlinson 1967) is an attempt to maintain a data base of Canada's resources. Information currently available in map form has been encoded by means of

Table 5.1

Some Signal Codes Used by Freeman

CODE	MEANING
0400	End of chain.
0404	Valid 04 chain combination.
0407UV	Serial Number. Immediately following this code is a serial number UV digits long.
0411	Non-chain data follows.
04127777	End of non-chain data. (Negates 0411 signal.)
0417UXYZ	Element-repeat code. The element U is repeated XYZ times.
0414UVWXZY	Rotation indicator. Indicates chain has been rotated by an angle of U.VWXZY radians.

a drum scanner for inclusion in the data base.

One of the most interesting features in the system is the use of the map "frame". With a large data base it is quite inconvenient to scan the entire base. If the map is divided into frames, and searches through the data base are keyed on the frame number, we may reduce the number of references to the data base. For example, if a map has 100 frames, and we wish to find some feature in the area of, say frame 55, we need not search all 100 frames to determine the closest features, but merely the frames that adjoin frame 55.

The data base is split into two data sets:

(1) the descriptor-data set (DDS) containing descriptive data;

(2) the image-data set (IDS) containing the outline data.

In the DDS for each map entity, there is a list of pointers to the frames containing relevant parts of the outline for that entity.

The fact that one type of data is not embedded within the other would enable retrieval of outline data for display purposes to be performed quickly. It is unfortunate, however, that the pointers to the various frames are contained within the DDS. This means that the display data cannot be used independently of the descriptive data. Storage overhead would be reduced if the descriptor-data sets were not required for producing display images.

5.5 SUMMARY

We have reviewed interesting aspects of three contrasting approaches to the problem of determining a computer representation of map data.

Cook has made lavish use of storage in order to describe explicitly the pictorial relationships between map elements. The resulting data structure, while it is relatively easy to understand, is unsuited to interactive use, due to the amount of processing required to reconstruct the outline of a map region.

Freeman has chosen an extremely simple structure that may be used efficiently for display purposes. However, by interspersing descriptive data with image data, access to the descriptive data becomes awkward and slow.

Tomlinson distinguishes between descriptive data and image data but access to the image data is only possible through the descriptive data. Tomlinson also has used "frames" to improve the response time for obtaining information from the data base.

In the next Chapter, the author presents his design for a data base of campus resources. Both the "frame" concept and the technique of separating descriptive and image data are utilized.

CHAPTER VI

A DATA BASE OF CAMPUS RESOURCES

6.1 INTRODUCTION

The implementation described in Chapter 2 allows for some input and evaluation of objectives, as described in Section 2.4. However, the simple data structure used did not allow searches over the data to be localized to one area of interest. In order to overcome this shortcoming and to allow a wider range of objectives to be input to the system, the author investigated the problem of defining a data structure that would allow both image data to be readily displayed and descriptive data to be used efficiently.

The model described was not used in the implementation of the Sieve Process at the University of Alberta, but has since been implemented (and studied) on a Burroughs B6718 at the University of Canterbury. The results of this study are reported in Section 6.5.

While several features may be unique to a given problem, many features (for example, BASE MAP, UTILITIES, and ROADWAYS), will be used for most problems. If a standard data base were developed, containing descriptive data and image data for all common features, then the work needed for any one problem would be greatly reduced. At the same time, by designing a structure which may be modified to contain features unique to particular problems, the problem of data base design may be solved once and for all.

A computer model of campus resources, including the data structure and data management routines, has been developed. The main design criteria for the model were those dictated by the intention to use the model in an interactive environment. While improvements are possible, use of the computer model takes us one step further towards a computer implementation of planning systems.

6.2 A DATA BASE FOR AN INTERACTIVE ENVIRONMENT - DESIGN OBJECTIVE

The data base will be used in an interactive environment. The data base should contain descriptive data for the purposes of evaluation, and image data for display on the CRT. In this section, we consider the major design criteria for such a data base.

A major problem with a data base designed for interactive use is that of obtaining efficient access to data. In an interactive graphics program, the function used most frequently is that required to display data on the CRT. This function must be performed efficiently if processing costs and response time are to be minimized. An important criterion, then, is that the data be structured in such a way that a display file may be constructed quickly.

The planner/programmer must have available a means of developing a modified model for a particular problem. One way to accomplish this is to create a copy of the original data base, and then modify this copy appropriately. The same data management routines should be useable for both the original and the copy. Thus an important criterion is that the data be structured in such a way that the user is able to modify (by addition or deletion) the information contained in his copy of the data base.

A third criterion, necessary if evaluation is to be made in an interactive environment, is that the data should be organized to allow searches through the data to be localized as much as possible.

In summary, there are three major design criteria:

(1) Both descriptive data and image data must be readily accessible for display.

(2) Data must be structured so that it can be easily modified.

(3) Data must be organized so that searches through the data are localized as much as possible.

6.3 THE DATA BASE

With the objectives summarized above, the author has developed a data structure for a computer data base of information on campus resources, and the related data management routines for retrieving, adding or modifying data. The data base is to contain information which allows any resource entity to be graphically illustrated, plus descriptive data unique to the type of resource considered.

To illustrate the data structure, we use a simple example. The map illustrated in Figure 6.1 is divided into four sectors (S_1, S_2, S_3 and S_4) and shows three features: roadways X, Y, open space* Z, and buildings V, W.

---

* By "open space" is meant an area such as a quadrangle.

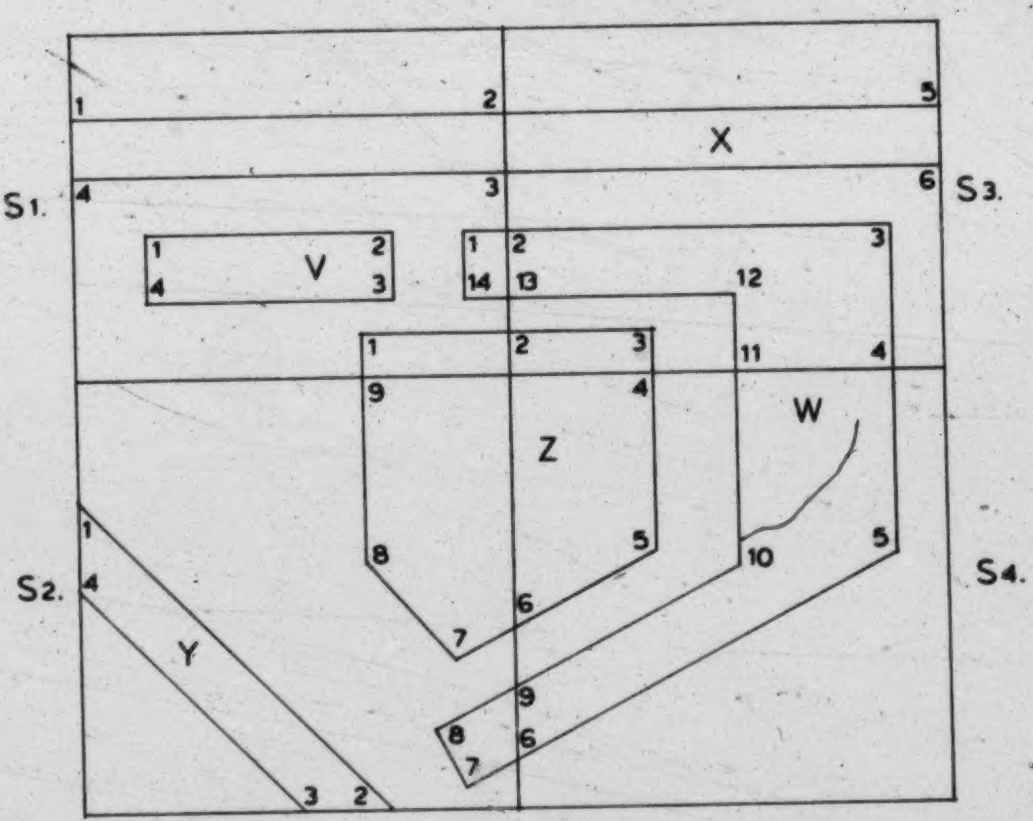

S1.
S2.
S3.
S4.
X
V
Z
W
Y
1
2
3
4
5
6
1
2
3
4
1
2
13
14
3
12
11
4
10
5
9
8
7
6
1
2
3
4
5
6
7
8
9
1
4
3
2

Figure 6.1 Sample Map Data

There are six block types (Figure 6.2). The function of each block type is as follows:

(1) MAP block - contains two tables of pointers to enable the structure to be accessed with respect to a particular sector, or a particular feature.

(2) SECTOR block - gives access to a table of pointers to the resource entities for any one feature located in any one sector.

(3) FEATURE SECTOR block - contains a table of pointers to the set of resource entities of one feature that occur in one sector.

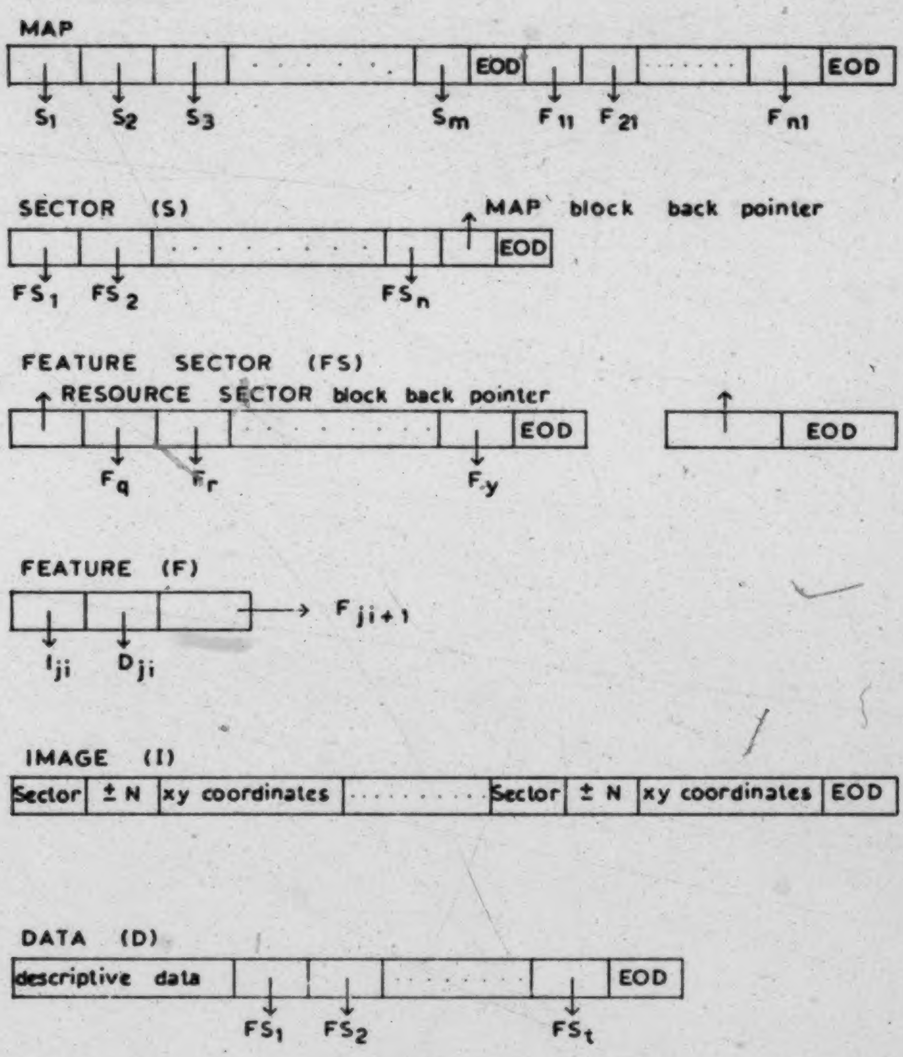
(4) FEATURE block - the descriptive data and image data for any resource entity are separate. Each FEATURE block in the list of FEATURE blocks for a feature points to the data for one resource entity of that feature.

(5) DATA block - contains descriptive data, and back-pointers to enable modification of the structure.

(6) IMAGE block - contains graphical data for display purposes.

Figure 6.3 shows the interrelationship of the different block types. Figure 6.4 shows each of the blocks described above in the data structure required for the example illustrated in Figure 6.1.

Let us now consider each of the block types in detail. In what follows, we represent the address of B (that is, a pointer to block B) in a field of a block by either


MAP
EOD
EOD
S_1
S_2
S_3
S_m
F_{11}
F_{21}
F_{n1}
SECTOR (S)
MAP block back pointer
EOD
FS_1
FS_2
FS_n
FEATURE SECTOR (FS)
RESOURCE SECTOR block back pointer
EOD
EOD
F_q
F_r
F_y
FEATURE (F)
F_{ji+1}
I_{ji}
D_{ji}
IMAGE (I)
Sector
± N
xy coordinates
Sector
± N
xy coordinates
EOD
DATA (D)
descriptive data
EOD
FS_1
FS_2
FS_t

BLOCK NAME	SIZE (IN WORDS)	NUMBER REQUIRED
Map	m+n+2	1
Sector	m+2	n
Feature Sector	variable	m×n
Feature	3	k
Image	variable	k
Data	variable	k

where:

k is the number of resource entities
m is the number of sectors
n is the number of features

Figure 6.2 Block Types

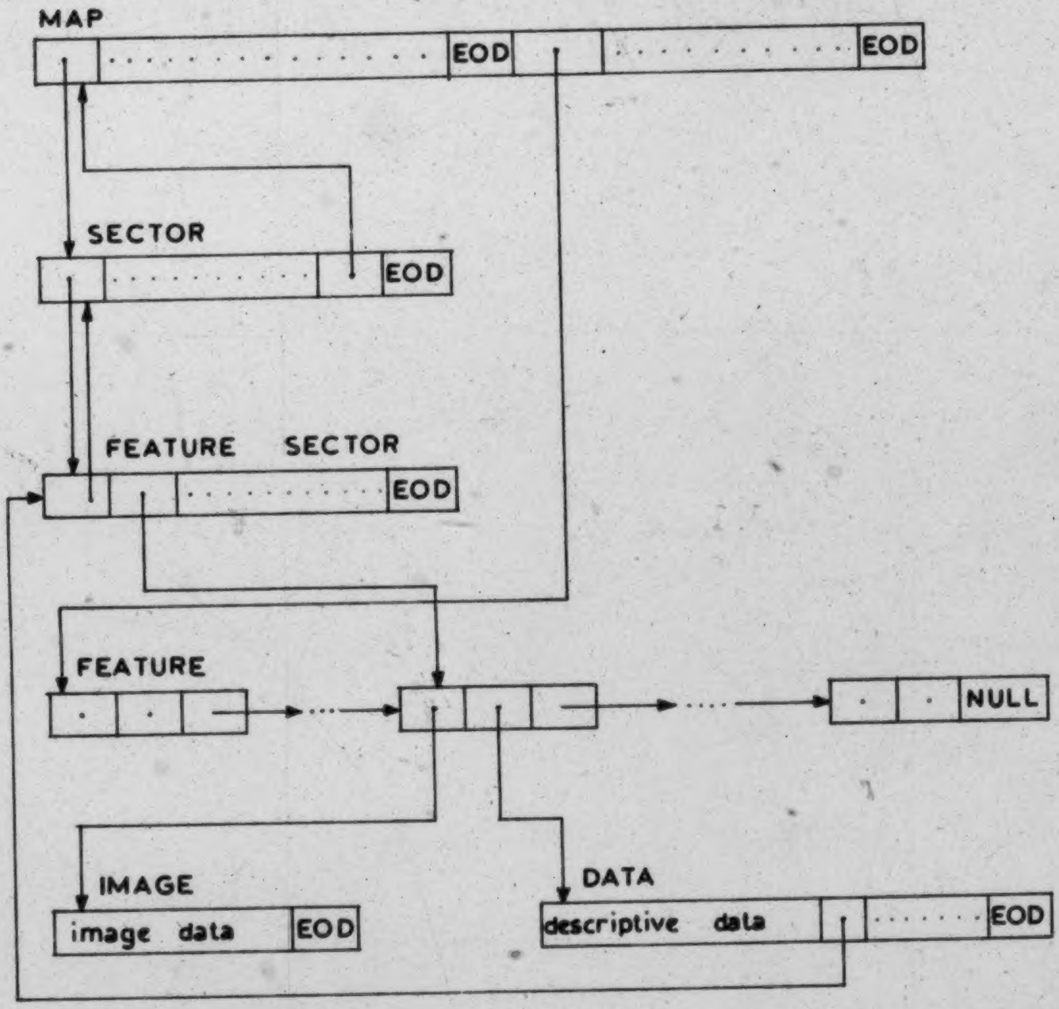

MAP
EOD
EOD
SECTOR
EOD
FEATURE SECTOR
EOD
FEATURE
NULL
IMAGE
image data
EOD
DATA
descriptive data
EOD

Figure 6.3 Macro Structure of Data Base

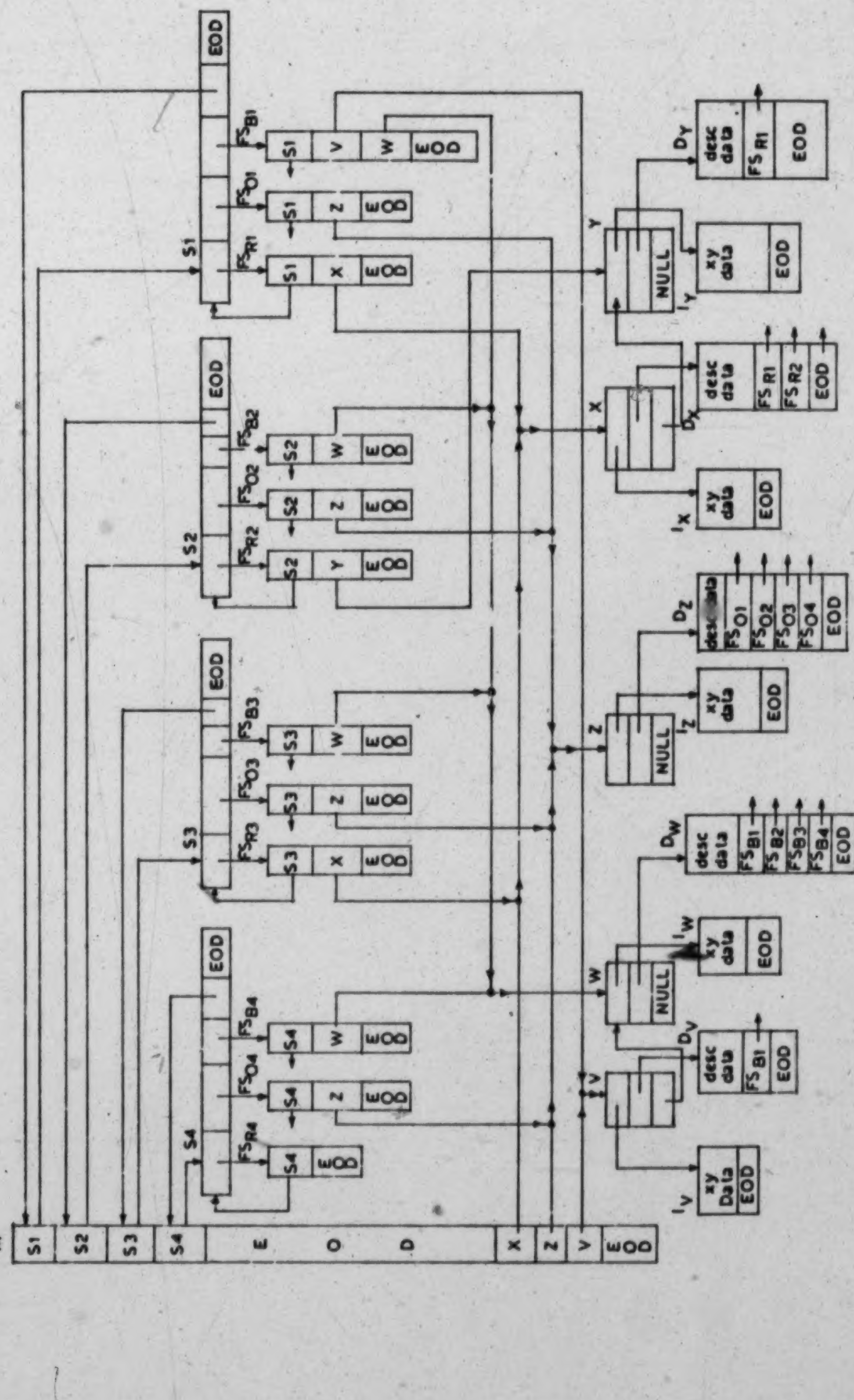

M
S1
S2
S3
S4
E O D
X
Z
V
EOD
FS_R1
FS_O1
FS_B1
FS_R2
FS_O2
FS_B2
FS_R3
FS_O3
FS_B3
FS_R4
FS_O4
FS_B4
W
Y
NULL
I_V
I_W
I_X
I_Y
I_Z
D_V
D_W
D_X
D_Y
D_Z
xy data
desc data

Figure 6.4 Data Structure for Map Illustrated in Figure 6.1

$\boxed{B}\rightarrow$ or A(B), depending on what gives the clearer picture of the structuring. The block titles (e.g. M, S_1, X, etc.) may be used to relate the individual blocks to the overall structure shown in Figure 6.4.

(1) MAP Block

One MAP block is used. The block contains a table of pointers to the SECTOR blocks for each sector, and a table of pointers to the initial element in each linked list of FEATURE blocks. Each of the tables terminates with an EOD (End Of Data) mark.

In the example (Figure 6.1), there are four sectors and three features. Thus it will require a MAP block which contains (4 + EOD + 3 + EOD) fields:

M

$A(S_1)$	$A(S_2)$	$A(S_3)$	$A(S_4)$	EOD	A(X)	A(Z)	A(V)	EOD

where S_i i=1,4 are SECTOR blocks and X, Z, V are the initial nodes on the list of FEATURE blocks for the individual features.

(2) SECTOR Block

The SECTOR block for any sector contains a table of pointers to FEATURE SECTOR blocks, one for each feature, a backpointer to the MAP block, and ends with an EOD mark.

There is one SECTOR block per sector, and for the example, four SECTOR blocks are required. In the example, there are three features, so each block will contain five fields:

S_1

$A(FS_{R1})$	$A(FS_{O1})$	$A(FS_{B1})$	A(M)*	EOD

* backpointers to the MAP block

S_2

$A(FS_{B2})$	$A(FS_{O2})$	$A(FS_{B2})$	A(M)+1*	EOD

S_3

$A(FS_{R3})$	$A(FS_{O3})$	$A(FS_{B3})$	A(M)+2*	EOD

S_4

$A(FS_{R4})$	$A(FS_{O4})$	$A(FS_{B4})$	A(M)+3*	EOD

where R, O, B stand for Roadways, Open Space, Buildings respectively. Thus FS_{R2}, for example, is the FEATURE SECTOR block for the Roadways feature and sector 2.

(3) FEATURE SECTOR Block

For any sector, there is one FEATURE SECTOR block for each feature. The block contains pointers to FEATURE blocks identifying the resource entities for the particular feature and sector under consideration. The block begins with a backpointer to the SECTOR block for the sector and ends with an EOD mark. If there is no resource entity for the feature and sector under consideration, the FEATURE SECTOR block contains only the backpointer and the EOD mark.

For the example, there are four sectors and three features, so we need 4x3=12 FEATURE SECTOR blocks:

* backpointers
to SECTOR
blocks

FS_{R1}

A(S_1)*	A(X)	EOD

FS_{R2}

A(S_2)*	A(Y)	EOD

FS_{R3}

A(S_3)*	A(X)	EOD

FS_{R4}

A(S_4)*	EOD

FS_{O1}

A(S_1)*	A(Z)	EOD

FS_{O2}

A(S_2)*	A(Z)	EOD

FS_{O3}

A(S_3)*	A(Z)	EOD

FS_{O4}

A(S_4)*	A(Z)	EOD

FS_{B1}

A(S_1)*	A(V)	A(W)	EOD

FS_{B2}

$A(S_2)$*	A(W)	EOD

FS_{B3}

$A(S_3)$*	A(W)	EOD

FS_{B4}

$A(S_4)$*	A(W)	EOD

(4) FEATURE Block

For each feature described in the data base there is a list of FEATURE blocks. A pointer to the first element in each list is contained in the MAP block. There is one FEATURE block for each resource entity. The block contains three pointers: a pointer to the IMAGE block describing the outline or graphical image of the resource entity; a pointer to the DATA block containing descriptive data for the entity under consideration; and a pointer to the next FEATURE block in the list. The last pointer is a NULL pointer for the last element in the list.

In the example there are two entities of the feature roadways, one entity of the feature open space and two of the feature buildings. Thus we need five FEATURE blocks, composed into three lists.

X

$A(I_X)$	$A(D_X)$	A(Y)

Y

$A(I_Y)$	$A(D_Y)$	NULL

Z

$A(I_Z)$	$A(D_Z)$	NULL

V

$A(I_V)$	$A(D_V)$	A(W)

W

$A(I_W)$	$A(D_W)$	NULL

(5) IMAGE Block

The IMAGE block contains the information required to reconstruct the outline of the resource entity for display on a CRT or plotter.

Each line segment in a sector is encoded as:

(1) sector number;

(2) ± number of data points,

- + if the interior of the region representing the resource entity is to the right of the line segment as it is traversed,
- - if the entity is to the left of the line segment;

(3) the XY coordinates for the segment contained within the sector.

Thus, for example, if the outline of a resource entity contains 3 segments, there will be three sets of the above data. The block definition is terminated with an EOD mark.

As there are five resource entities in the example, we need five IMAGE blocks. Each of these will contain the necessary information required to reconstruct the outline of the entity on the CRT.

The IMAGE blocks for X and Y are encoded below. The X and Y coordinates are those from the numbered points on the outlines in Figure 6.1.

I_X

1	2	X1	Y1	X2	Y2	1	2	X3	Y3
X4	Y4	3	-2	X5	Y5	X2	Y2	3	2
X6	Y6	X3	Y3	EOD					

I_W

1	-4	X2	Y2	X1	Y1	X14	Y14	X13	Y13
3	3	X2	Y2	X3	Y3	X4	Y4	3	-3
X13	Y13	X12	Y12	X11	Y11	4	3	X4	Y4
X5	Y5	X6	Y6	4	3	X9	Y9	X10	Y10
X11	Y11	2	4	X6	Y6	X7	Y7	X8	Y8
X9	Y9	EOD							

(6) DATA Block

The DATA block for any resource entity contains descriptive data relating to the feature under consideration. The block is divided into two sections:

(1) data regarding the resource entity;

(2) backpointers to FEATURE SECTOR blocks.

The second portion is necessary for proper updating of the structure if a resource entity is deleted. It is a variable length list of backpointers to those elements in FEATURE SECTOR blocks that point to the FEATURE block that points to this block. The block definition is terminated with an EOD mark.

For the example, we have five resource entities. Thus, five DATA blocks are required:

D_X

Descriptive Data	A(FS_{R1})	A(FS_{R3})	EOD

D_Y

Descriptive Data	A(FS_{R2})	EOD

D_Z

Descriptive Data	A(FS_{O1})	A(FS_{O2})	A(FS_{O3})	A(FS_{O4})	EOD

D_V

Descriptive Data	A(FS_{B1})	EOD

D_W

Descriptive Data	A(FS_{B1})	A(FS_{B2})	A(FS_{B3})	A(FS_{B4})	EOD

Descriptive data for the feature Roadways might include:

(1) The name of the road.

(2) An indication whether the road is:
an emergency vehicle access route;
paved or gravel.

Descriptive data for the feature Open Spaces might include:

(1) Area.
(2) A reference point (centre of mass).
(3) Access streets or buildings.
(4) Historical Importance.

Descriptive data for the feature Buildings might include:

(1) The name of the building.
(2) Area.
(3) Number of classrooms.
(4) Major entrances.

6.4 IMPLEMENTATION

While it must be updated periodically, the data base, if it is to be used for more than one application, must be made available to the general user as READ ONLY data. Each user, however, may use the data base management routines to develop his own version of the data base.

When the user requires use of the model, he is given a copy of the MAP, SECTOR, FEATURE SECTOR, and FEATURE blocks. Any attempt to modify an IMAGE or DATA block results in a copy of the referenced block being made available in core for modification. Modifying pointers in any block results in modification of only the user's copy. Naturally, the user has the ability to preserve his modified copy from session to session.

Persons responsible for the data base may, operating in privileged mode, use the management routines to modify the original data base when necessary.

The applications programmer requires access to the data base by feature, sector, and resource entity. Access to data by feature is possible through the use of the table of pointers in the MAP block. Access to data by sector is achieved in the same manner. Access to the descriptive data or image data is by use of the FEATURE blocks for the feature being considered.

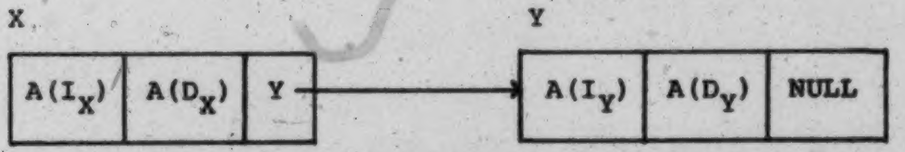
In the following discussion we shall refer to the "feature index" (FINDEX) as the index of a feature in the table of features in the MAP block. For example, in Section 6.3 we specified the MAP block M

$A(S_1)$	$A(S_2)$	$A(S_3)$	$A(S_4)$	EOD	A(X)	A(Z)	A(V)	EOD

The features Roadways, Open Space and Buildings would have FINDEX numbers 1, 2 and 3 respectively. SINDEX, the "sector index", is similarly defined.

We define the "resource entity index" (RINDEX) as the index of the FEATURE block for the resource entity in the list of FEATURE blocks.

For example, in Section 5.4 we have the blocks



Thus, the resource entities X and Y would have RINDEX values 1 and 2 respectively. Similarly, Z, V and W would have RINDEX values 1, 1 and 2 respectively.

While the programmer may define his own management system, there are available for his use the following data management procedures:

(1) COPY (NFILE)

A copy of the data base, preserved on the file NFILE by means of the RETAIN procedure, is copied into core.

(2) RETAIN (MPOINT,NFILE)

A copy of the data base is preserved on the file NFILE. MPOINT is a pointer to the MAP block for the data base.

(3) LOCATE (FINDEX,RINDEX,DPOINT)

DPOINT is returned as a pointer to the FEATURE block of the RINDEX resource entity of the FINDEX feature.

(4) MODIFY (FINDEX,RINDEX,DORI,NEWDATA)

The data for the RINDEX resource entity of the FINDEX feature is to be replaced. DORI is a flag to indicate whether descriptive or image data is to be modified. NEWDATA is the updated version of the data to be inserted.

(5) ADDRE (FINDEX,DDATA,IDATA)

A new resource entity is added to the list of resource entities for the FINDEX feature. This means a FEATURE block is created and added to the end of the appropriate list. DDATA and IDATA are arrays holding the descriptive data and image data respectively.

(6) ADDF (DDATSIZ)

A new feature is added to the list of features of the MAP block. DDATSIZ is the size of the descriptor of the feature.

(7) ADDS (SINDEX)

A new sector is inserted in the list of sectors in the MAP block at the position SINDEX or at the end of the list, whichever occurs first.

(8) DELETE (RINDEX,FINDEX)

The RINDEX resource entity of the FINDEX feature is deleted from the data base.

(9) DELETF (FINDEX)

The FINDEX feature is deleted from the data base; that is, all related resource entities are deleted and the list of features in the MAP block is shortened by one.

(10) DELETS (SINDEX)

The SINDEX sector is deleted from the data base. All image data associated with the sector is deleted. The SECTOR block and all related FEATURE SECTOR blocks for the sector are deleted.

There are three global variables MPOINT,NFEAT and NSECT. MPOINT is a pointer to the current MAP block. NFEAT is the number of features defined. NSECT is the number of sectors defined.

The procedures available do not cover the full range of possible requirements. For example, there is no procedure to display (or plot) image data. The data base has been designed to be graphics hardware-independent. At the time of its design, no decision had been made as to the type of terminal to be installed at the University of Canterbury. (At time of writing, a DEC-GT44 has been ordered.) However, the requirement to have a data base which would be usable with whatever terminal was finally chosen is an advantage in that the data base is more easily converted for use on other systems. Plotted output may be obtained by using the LOCATE procedure to locate data for the resource entity and inputting the image data to the proper plot routines.

6.5 DISCUSSION

Let us now consider how well the data structure satisfies the three design criteria set out in Section 6.2. The data, both image and descriptive, for any resource entity is readily obtainable. From the table of feature pointers in the MAP block, the programmer may obtain a pointer to the initial element in the linked list of FEATURE blocks for the feature. By one more level of indirect addressing, the data, either descriptive or image, is available (Figure 6.5). Alternatively, by four levels of indirect addressing, the programmer may access any data for resource entities subdivided into both sectors and features (Figure 6.6).

The structure is easily modified. Procedures are available which may be used in a straightforward manner to modify the data for any resource entity. Procedures are also available to add or delete features or sectors.

To localize searches through the data base, the author has used the notion of sectors proposed by Tomlinson. By

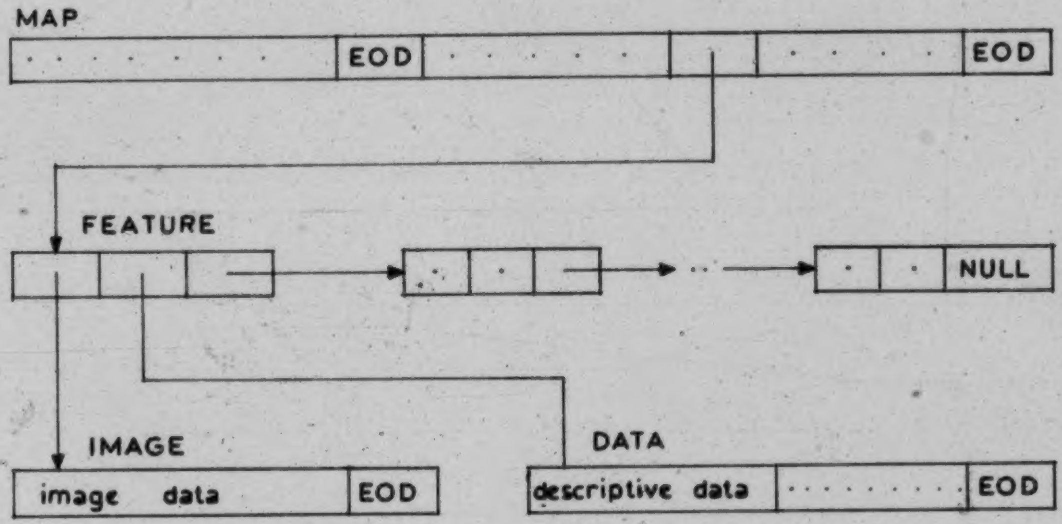

MAP
EOD
EOD
FEATURE
NULL
IMAGE
image data
EOD
DATA
descriptive data
EOD

Figure 6.5 Access of Data by Feature

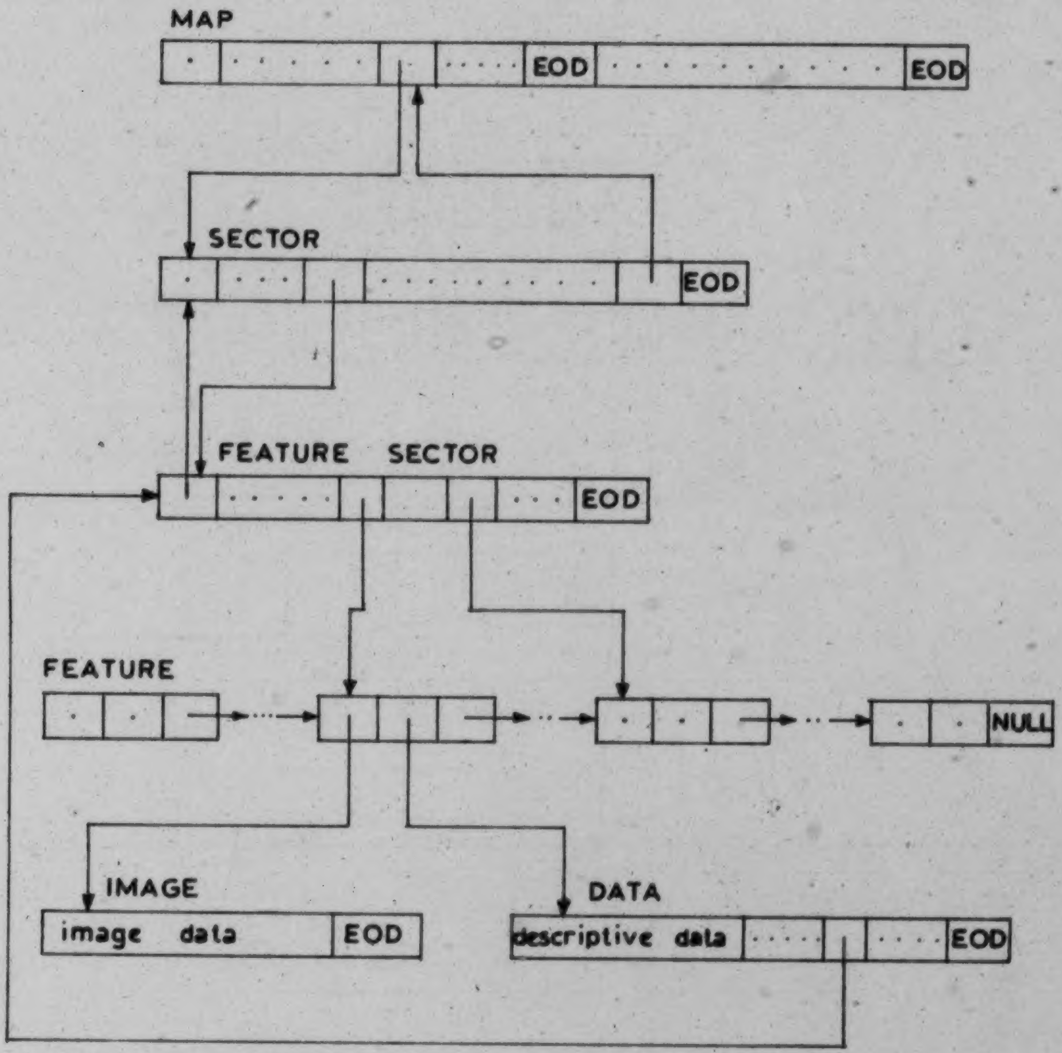

MAP
EOD
EOD
SECTOR
EOD
FEATURE SECTOR
EOD
FEATURE
NULL
IMAGE
image data
EOD
DATA
descriptive data
EOD

Figure 6.6 Access of Data by Sector

associating a site with a sector, searches through the data base may be restricted to adjacent sectors.

One method by which this can be done is to assume that the map under analysis is overlaid with a matrix of points, so that each point (i,j) represents one sector. An element, P, on the map in sector k, may be related to the point (i_k, j_k). By incrementing or decrementing i_k and j_k we may obtain a list of all sectors surrounding sector k. By a process analogous to the way FORTRAN array elements are mapped on to storage locations, we may map any point (i,j) onto the table of SECTOR pointers in the MAP block. Thus, when searching for the presence of a resource entity of a particular feature we may focus our search on adjoining sectors, working out from the centre point rather than considering the entire list of resource entities for the feature desired.

Use of sectors also has an advantage when windowing the image data for display purposes, in that we may delimit the sectors visible on the screen at any one time and consider data only from those sectors.

6.6 SUMMARY

A data structure for a computer model of campus resources has been developed for use in an interactive graphics environment. The data base is to contain information that is required for a wide range of planning problems. For any particular problem, a modified copy of the data base, including data required for the project, is obtainable in a straightforward manner.

CHAPTER VII

INTERACTIVE GRAPHICS PROGRAMMING

7.1 INTRODUCTION

"The most successful uses of computer graphics are simple applications and demonstrations" (SIGGRAPH 1969). The difficulty is that:

(1) graphical communication is intrinsically complex;

(2) graphics software support is inadequate.

The considerable time needed to develop the map storage and retrieval application led the author to study several aspects of interactive graphics programming, with the aim of developing more useful software.

In this chapter, methods of communication between an application program and the user at the CRT are discussed. The discussion which follows in this chapter and in Chapters 8 and 9, assumes that the hardware configuration comprises an "intelligent terminal" (Machover 1969) linked with a much larger CPU. Such a terminal is usually a composite of a CRT with a (small) processor.

These processors may be very limited in scope (e.g. the CDC-160A used with the GRID) or a more powerful computer, such as a PDP-9 used in the system described by Newman (1969).

Two major problems common in all interactive graphics programs have been considered:

(1) the definition of the Console Command Language (CCL);

(2) the implementation of an interpreter for the CCL.

Every interactive program must provide (implicitly if not explicitly) a command language for its control. Newman and Sproull (1973) note that the "choice of a command language will often play a significant part in determining the success or failure of the program. It is therefore surprising to find that, despite the importance of command languages, relatively little has been done to make these languages easier to design and implement."

As a result of the author's investigation, a new graphics interface between the user at the CRT and the application program has been developed. The definition of elements in a CCL has been formalized. A CCL interpreter for use with (any) interactive graphics programs has been developed. A major improvement over software previously available is that the programming requirements are modularised, that is, the programmer provides modules for each command in the CCL independently. By removing the interpreter from the application program, we remove what is a major portion of most interactive graphics programs, and, since graphical communication is complex, what is often the most difficult programming effort required for interactive graphics applications. This new support system is outlined in Chapter 8, with discussion in Chapter 9.

In this chapter we have followed definitions given by Joyce and Cianciolo (1967). A user is "a person who manipulates console controls.....to do productive functions such as mechanical or electrical design or parts layout. This person usually has no knowledge of computer hardware or software and is only interested in the console as a tool to get his own job done."

An (applications) programmer is "the person who provides the software to do specific productive jobs."

7.2 COMMUNICATION THROUGH A GRAPHICS TERMINAL

A graphics user communicates with the machine via a Lightpen, Function Keys and Status Switches, or an Alphanumeric Keyboard. Some facilities have a Rand Tablet (Davis and Ellis 1964) or equivalent device for input of vectors.

The display of a "logic gate" node on the screen is a symbolic representation of a "logic gate" node in the data structure of a computer model. Zeros and crosses could just as easily (though inappropriately) be used instead of the conventional images to represent logic gates pictorially. The picking with a lightpen of a "symbolic entity" on the screen is related to that portion of the display model which defines the entity, and a mapping is used to determine which part of the problem model has been identified. This notion of symbolic representation of display models is important, for it forms the basis of most graphic data-structure systems. In many instances the lightpen is replaced with a joystick, a set of cursors or a "mouse".

Function keys (with a variety of status switches) are also available to the user. Generally they are used not to specify parts of the computer model, but rather to identify which data manipulation function is to be applied to a part of the model that has been indicated by picking with the lightpen. With programmable terminals such as the GRID, some of the function keys may be used to control the resident supervisor.

The alphanumeric keyboard is used mainly for:

(1) labelling diagrams or naming files;

(2) inserting numeric values.

With the lightpen, function keys and an alphanumeric keyboard, the user should be able to communicate with the machine in a manner consistent with the knowledge and experience of the user.

The applications programmer must define a Problem Oriented Language for use between man and machine. The language consists of commands to the system or requests by the user for information resident in the system. Since it is for use at a graphics console, we call the language a Console Command Language (CCL).

There are several desirable attributes for this type of language. Among these are that:

(1) the language be easy to learn and remember;

(2) each command be relatively simple and not composed of an excessive number of separate components;

(3) the language be of a form that is convenient for the person for whose use it is intended.

The program processing this language should be such that:

(1) new commands may be added easily;

(2) any commands may be modified.

7.3 SOFTWARE FOR INTERACTION BETWEEN THE USER AND THE APPLICATION PROGRAM

Machover (1969) defines an "intelligent terminal" as one which can handle the following tasks without support from the central computer:

(1) Generate a steady nonflickering display.

(2) Generate typical graphical elements such as characters, lines, etc.

(3) Accommodate common display "housekeeping" tasks such as lightpen tracking, alphanumeric keyboard message composition and editing, picture scaling, moving and copying.

Among the devices which qualify as intelligent terminals are PDP 338, Graphic II, IBM 2250 Mod 4, and GRID.

Given the assumption that the CRT facility is an intelligent terminal, there have been two distinct approaches towards the implementation of a graphics software support facility:

(1) The small machine executes much of the user's program, with program segments where possible being automatically loaded from the main machine. Larger program segments may be executed in the main machine (Newman 1968, 1969), (Boullier et al 1972).

(2) The small machine acts as a combined message switcher and data compression device (Morrison 1967), (Cotton and Greatorex 1968), (Huen 1969).

The two approaches differ in the way in which the user-machine interface is defined. In the first approach, for each user input there is an appropriate module or program segment executed, usually, by the small machine. Newman (1968, 1969) points out that this interaction may be effectively represented by a State Diagram. In the second case, user inputs are collated and transmitted to the main machine. For each "Report Block", describing a sequence of user actions rather than just one action, there is a module or program to be executed by the main machine. CCLs used with this type of interface will tend to be phrase-structure grammar oriented.

Illustrations of the two approaches are given in the following sections.

7.3.1 A STATE-DIAGRAM REPRESENTATION OF THE INTERFACE

An ACTION is an input, which may produce a response; the corresponding REACTION defines this response, and in addition, any unmanifested effects of the action on the state of the machine.

Let us consider an example given by Newman (1968), that of drawing a "rubber-band" line. The line "can be created with the aid of a lightpen and one push button, in a sequence of five operations:

(1) press button to start pen tracking;
(2) track with pen to starting point of line;
(3) press button of fix starting point;
(4) track with pen to end point;
(5) press button to fix end point and stop tracking."

Figure 7.1a "shows a state diagram representing this

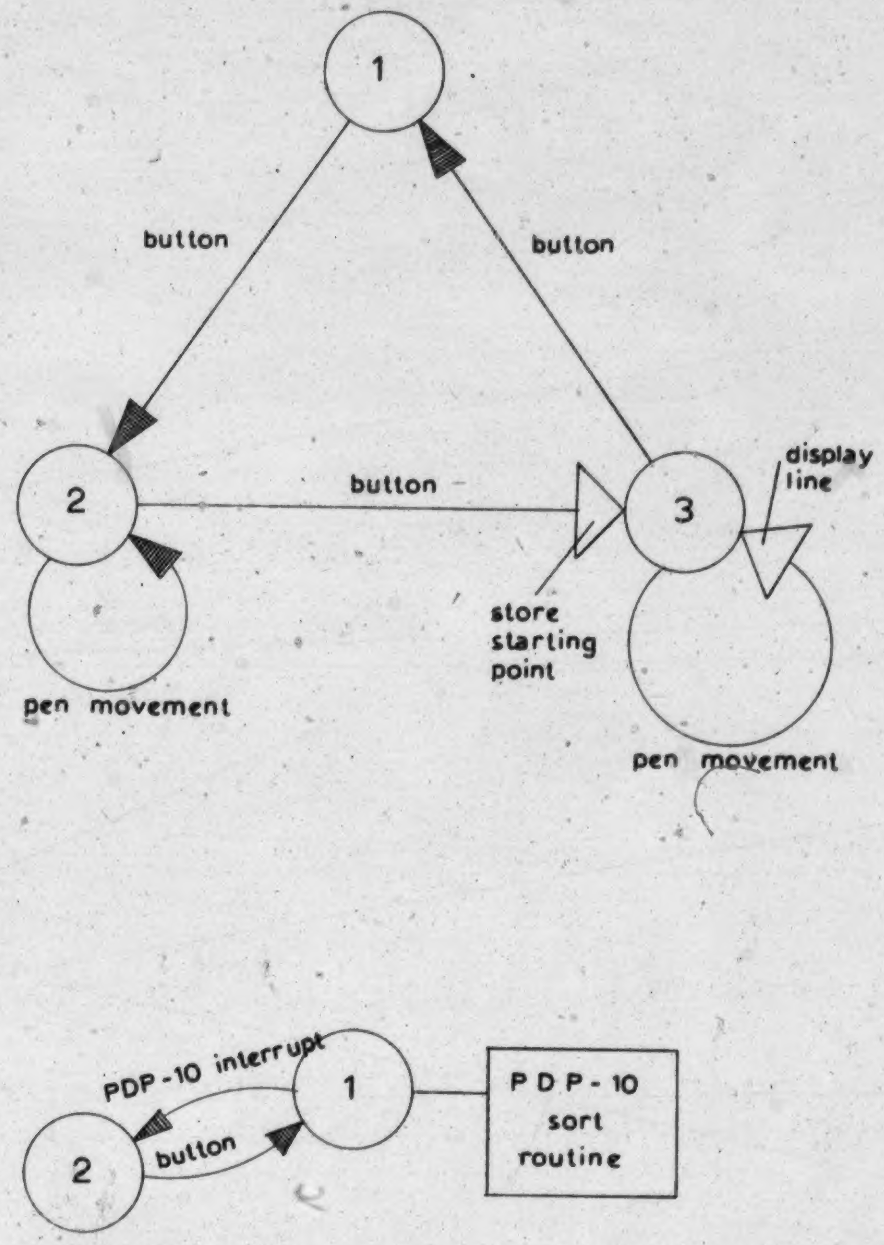

1
button
button
display
line
2
button
3
store
starting
point
pen movement
pen movement
PDP-10 interrupt
1
PDP-10
sort
routine
2
button

Figure 7.1 State-Diagram Representation of the Man-Machine Interface:

(a) Rubber-Band Line-Drawing by use of Terminal Processor.

(b) Interaction with the Main Machine (PDP-10).

sequence. Each branch represents an action, and the resultant reaction is specified in the 'arrowhead'. Only valid actions are included......... The inclusion or exclusion of an action may add semantic properties to the diagram. This is shown by the 'pen movement' branches on states 2 and 3, which imply pen tracking during those states and make explicit reference to tracking unnecessary."

In Figure 7.1b the box attached to the state symbol indicates that while the program is in that state a procedure is being executed in the main machine.

Newman remarks that: "A particular advantage of this technique is the way an immediate reaction can be associated with each action in a sequence; this is of great importance in graphical programs. On the other hand the state-diagram offers no direct method of attaching semantic functions to groups of actions and is therefore of little use for describing phrase-structured grammars. This is less of a drawback than it seems. An interactive problem-oriented language need not possess a complex structure to function efficiently, and benefit can often be gained from simplifying the language as much as possible." Among the beneficial aspects of this type of interface are that:

(1) There may be an "immediate" reaction to user actions.

(2) An invalid action may be detected as soon as it occurs.

(3) The user-program interface is simple to understand and may therefore be readily used by programmers.

There are several drawbacks to this type of interface, among the most serious being that:

(1) The programmer must decide whether modules are to be executed on the main machine or the small machine, and this obviously complicates the interface between the two machines.

(2) The programmer must write programs for two different systems.

(3) Since either machine may execute a user program, the data structure must be available to both machines and any modification to the structure by either machine must be communicated to the other machine. This implies much interaction between the two machines.

Newman and Sproull note that there is a tendency for programmers to write programs with large numbers of states. As a result, "the state-diagram technique may produce very complex programs defining quite simple command languages....... Since the program responds differently to each state, the user is likely to become confused unless he is clearly guided by means of direction from the program."

7.3.2 REPORT BLOCKS

The small machine may contain a standard supervisor which monitors user actions and makes temporary modifications to the display file. All changes made are strictly local to the small machine and have no effect on any structure in the main machine. Information concerning the user actions and display file modifications is retained in a "Report Block" which is transmitted to the main machine whenever the user requests that the block be sent. This information is made available to the application program in a general parametric form from which the program may reconstruct the actions of the console user and alter the data structure accordingly.

Among the major benefits of this form of interface are that:

(1) There is a standard interface between the two machines. This interface is controlled by the graphics supervisor, rather than by the user-program.

(2) The programmer has to write programs for only one machine.

(3) An "immediate" response to each input may be given as the input is added to the record block.

Drawbacks of this type of interface include that:

(1) An error in any element in the block usually requires the entire block to be re-input.

(2) Response time at the CRT is increased, since the CRT is generally not a "privileged" user of the main machine. In addition, a low speed communications line is often used to connect the two machines.

7.4 DISCUSSION - DESIGN OBJECTIVES FOR A MAN-MACHINE SOFTWARE INTERFACE

Every interactive program contains a definition of the program CCL. The CCL may be explicit as some form of table or may be implicit in flow of control through the application program. If the latter is the case, the addition of new commands or modification of existing commands is very complicated. These problems are avoided by using a separate table-driven command processor. To implement a new CCL or modify an existing one, only changes to tables are needed.

Newman and Sproull note that CCLs are generally best defined by using some form of table-driven language. They use the term "control-oriented language" to describe a language used to define a CCL in tabular form. They conclude that the main difficulty in using control-oriented languages stems from the use of multiple input devices, since there is the problem of determining from which device or devices the command originated.

If we are to avoid making the application programmer learn yet another language, the control-oriented language should be either the same language that the programmer is using for the application program, or one quite similar to it.

When using an intelligent terminal, it is clear that a portion of the data base must be shared between the two processors. The main machine requires the data base. The small machine needs at least a portion of it for purposes of input and output. A basic problem in the definition of support systems for intelligent terminals is to decide which segments of the data base are required in the small machine. If there were a standard interface between the two machines then the support software may handle all tasks related to the management of sharing the data base. To the extent that this is possible, the application program will become easier to program and the CCL easier to modify after the initial implementation is completed.

In summary, the author sees the following design criteria as essential for the interface between user and application program:

(1) There should be, at all times, a standard interface between the two machines.

(2) There should be an immediate reaction to any user input, and an invalid action should be detected immediately it occurs.

(3) The programmer should program in only one language for what appears to be one machine. That is, the division of programs between the terminal and the main machine should be transparent to the programmer, and there should not be a separate language to define the interface.

(4) Above all, the software interface should be simple to understand and use.

In Chapter 8, the author describes an implementation of a graphics man-machine interface of the Report Block type. A discussion of the merits of the interface is contained in Chapter 9.

CHAPTER VIII

A TABLE-DRIVEN INTERPRETER FOR GRAPHICS APPLICATIONS

8.1 INTRODUCTION

In this chapter a table-driven interpreter to be used as a software interface between the user at the CRT and an interactive graphics application program is described. For this interpreter, the programmer of a given application is expected to:

(1) Supply, in table form, a definition of his Console Command Language. This definition includes a specification of the allowable operator actions (as terminal symbols), the allowable commands, and a mapping of commands on to subroutine names and error messages.

(2) Write the subroutines to be executed upon successful decoding of commands by the interpreter.

The general structure of an applications program is as shown in Figure 8.1. The command language interpreter and its associated preprocessors are delimited in the top lefthand side of the diagram.

A fundamental aspect of the method of language definition is the inclusion of a "lexicon" of allowable operator actions (Section 8.2.1). Actions are queued in the terminal until the "carriage return" key (taken as "end of message" (EOM)) is pressed. The input file of user actions is then transmitted to the interpreter. The syntax analyzer, to be described in Section 8.3, determines which (if any) command has been input

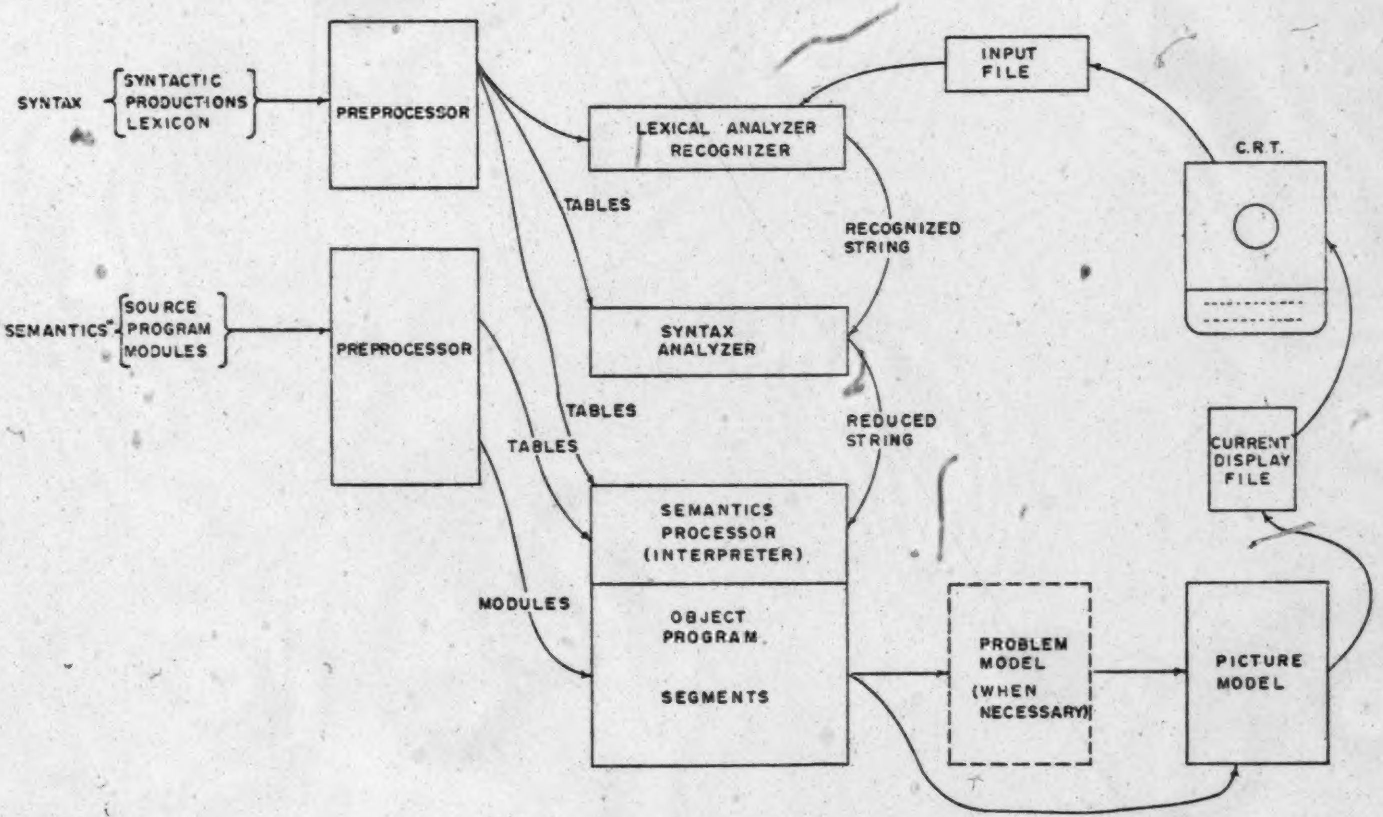

SYNTAX
SYNTACTIC
PRODUCTIONS
LEXICON
PREPROCESSOR
"SEMANTICS"
SOURCE
PROGRAM
MODULES
PREPROCESSOR
TABLES
TABLES
TABLES
MODULES
LEXICAL ANALYZER
RECOGNIZER
SYNTAX
ANALYZER
SEMANTICS
PROCESSOR
(INTERPRETER)
OBJECT
PROGRAM
SEGMENTS
RECOGNIZED
STRING
REDUCED
STRING
INPUT
FILE
C.R.T.
CURRENT
DISPLAY
FILE
PROBLEM
MODEL
(WHEN
NECESSARY)
PICTURE
MODEL

Figure 8.1 Man-Machine Interface Schematic

to the system and schedules the appropriate user program modules for execution. If the input command is syntactically inadmissable, the appropriate error message is displayed on the CRT.

The system has the advantages that:

(1) It eliminates the need for the programmer to code components of the application program to handle interaction with the user at the terminal. Instead, the programmer provides tables, a task which is a great deal simpler.

(2) The application program developed is entirely modular, and each module may be developed independently.

8.2 CONSOLE COMMAND LANGUAGE DEFINITION

The complete definition of a CCL comprises:

(1) A definition of the actions which an operator may take at the console; these "actions" are the primitive elements of the language.

(2) A definition of the syntax of the language, and a mapping of the syntax against names of procedures (to be executed if the message input by the user is syntactically correct) and error messages (to be displayed if the message includes errors).

8.2.1 LEXICAL DESCRIPTION OF USER ACTIONS

For most computer languages, the primitive elements are typed or punched characters. For a CCL, we consider the primitive elements to be *actions* taken by the console operator.

Each statement of a console language is regarded as a sequence of *components*, each of a specific *component type*. A component may correspond to a single action, or to a sequence of actions of the same class. Associated with each component type is a set of attention variables, global variables whose values are affected by input of a component of this type. An attention variable common to all types is *atct*, the component type variable.

For GRID, five component types* have been defined:

(1) Pen Pick
(2) Function Key
(3) Point String
(4) Vector String
(5) Character String.

The distinction between *actions* and *components* can be illustrated as follows: The drawing (picking) with the light-pen of 5 points on the screen requires 5 *actions* by the user. These actions may all be described as one *component* of a command.

Strings (of points, vectors or characters) are specified rather than single elements, because it is simpler and more usual for a programmer to refer to a complete contour or line of text. The attention variables associated with each component type (see Table 8.1) are made available to the

*While the choice of types is hardware-dependent, the essential idea should be adaptable for any hardware.

Table 8.1

Component Types and the Associated Attention Variables

Component Type	Attention Variables	#	Value of Variable Specifies
Pen pick	*atct*=1	0	Component type - pen pick.
	atbn	1	(Programmer-defined) block number of picked entity.
	atid	2	(Programmer-defined) identifier given to a copy of a block, or to part of a block.
	attype	3	Type (point, vector, character) of entity picked.
	atxp	4	Coordinates of picked entity.
	atyp	5	
	atcode	7	(If character picked), EBCDIC code.
	atctype	8	(If character picked), type of character - digit, letter, etc.
Function key	*atct*=2	0	Component type - function key.
	atfk	9	Value of function key pressed.
Point string	*atct*=3	0	Component type - point string.
	atxp	4	Coordinates of most recent point.
	atyp	5	
	ax	-	Arrays in which all coordinates are set to be accessible to program.
	ay	-	
	atn	6	Count of number of points in point string.

Table 8.1 (continued)

Component Type	Attention Variables	#	Value of Variable Specifies
Vector	*atct*=4	0	Component type - vector string.
	atxa	10	Coordinates of first end-point of most recent vector.
	atya	11	
	atxb	12	Coordinates of second end-point of most recent vector.
	atyb	13	
	ax	-	Arrays in which all coordinates are set.
	ay	-	
	atn	6	Count of number of vectors.
Character string	*atct*=5	0	Component type - character string.
	atxc	14	Coordinates of first character input.
	atyc	15	
	atxrc	16	Coordinates of most recent character input.
	atyrc	17	
	atcode	7	EBCDIC code of most recent character input.
	atctype	8	Type of character.
	achar	-	Array in which characters are accessible to the applications program.
	atn	6	Count of number of characters.

Table 8.1 (continued)

Notes

(1) The array names, *ax*, *ay*, *achar*, cannot be used in lexical definitions. All other attention variables are of type integer.

(2) In addition to the above component types, there are two special cases of character string: integer (*atint*) and real (*atreal*). These facilitate the input of numbers from the keyboard.

(3) An input action of any type can alter the values only of those variables associated with that type. If the action is rejected, the values of all variables remain unchanged.

(4) The attention variable *ats* is associated with all components and monitors the current setting of the status switches.

programmer as global variables, for example, in labelled COMMON for FORTRAN.

When writing productions of console language grammars any convenient symbols may be used as terminal symbols, with each symbol representing a statement component. The symbol is defined by a Boolean term involving ranges of values for the attention variables associated with the component type. Not all attention variables need to be used in a definition. Suppose, for example, that we wish to give the same interpretation to either the picking of a word (say, DELETE, defined as block number 10) in a menu or pressing of a function key (say function key 1). We might define the actions as follows:

DELETE ::= $(atct=1 \wedge atbn=10) \vee (atct=2 \wedge atfk=1)$ #

where # delimits the definition. Note that, in the lexical definition, the symbol DELETE stands for the actions of picking the word or pressing the function key. While it is appropriate to use the same symbol that appears on the screen, any other symbol could have been chosen, for example

PICKDEL ::= $(atct=1 \wedge atbn=10) \vee (atct=2 \wedge atfk=1)$ #

The symbol used in the lexical definition must of course be used in the syntactic definition.

The user may define each terminal symbol (i.e. each permissable action or set of actions) in the CCL in a similar manner, building a table of the form

Name_i ::= Term_i for i=1,n

where there are n terminal symbols in the CCL. This table constitutes a lexicon of allowable operator actions.

The method of lexical description appear to have considerable power. For example, the Sieve Process command <link> has the component <line> which may contain up to 24 vector segments, all of which must be within the map display portion of the CRT display. We can define LINE in the lexicon:

$$\begin{aligned}\text{LINE} ::= {} & atct=4\wedge atn\leq 25\wedge((atxa<850\wedge atya<850)\wedge \\ & (atxb<850\wedge atyb<850))\ \#\end{aligned}$$

The bracketting is optional, but is included for clarity.

8.2.2 SYNTAX AND SEMANTIC MAPPINGS

The author's experience has been that many interactive graphics application programs have less than twenty distinct user commands. These commands usually have few components (typically less than 5) with less than 50 entries in the lexicon. For example, Johnson's (1969) CALD program (illustrated in Figure 8.2) has 13 commands, the longest command has 4 components, there are 23 required entries in the lexicon. Because of this relative simplicity, due in part to the powerful method of lexical definition, the author has decided that languages should be specified to the interpreter as a tabulation of commands.

Given that the terminal symbols of the language have been completely defined in the lexicon, each command is defined as a sequence of triplets of the form

$$(\text{name}_i,\ \text{error}_i,\ \text{module}_i)$$

where

name_i - is the terminal symbol for the i^{th} component

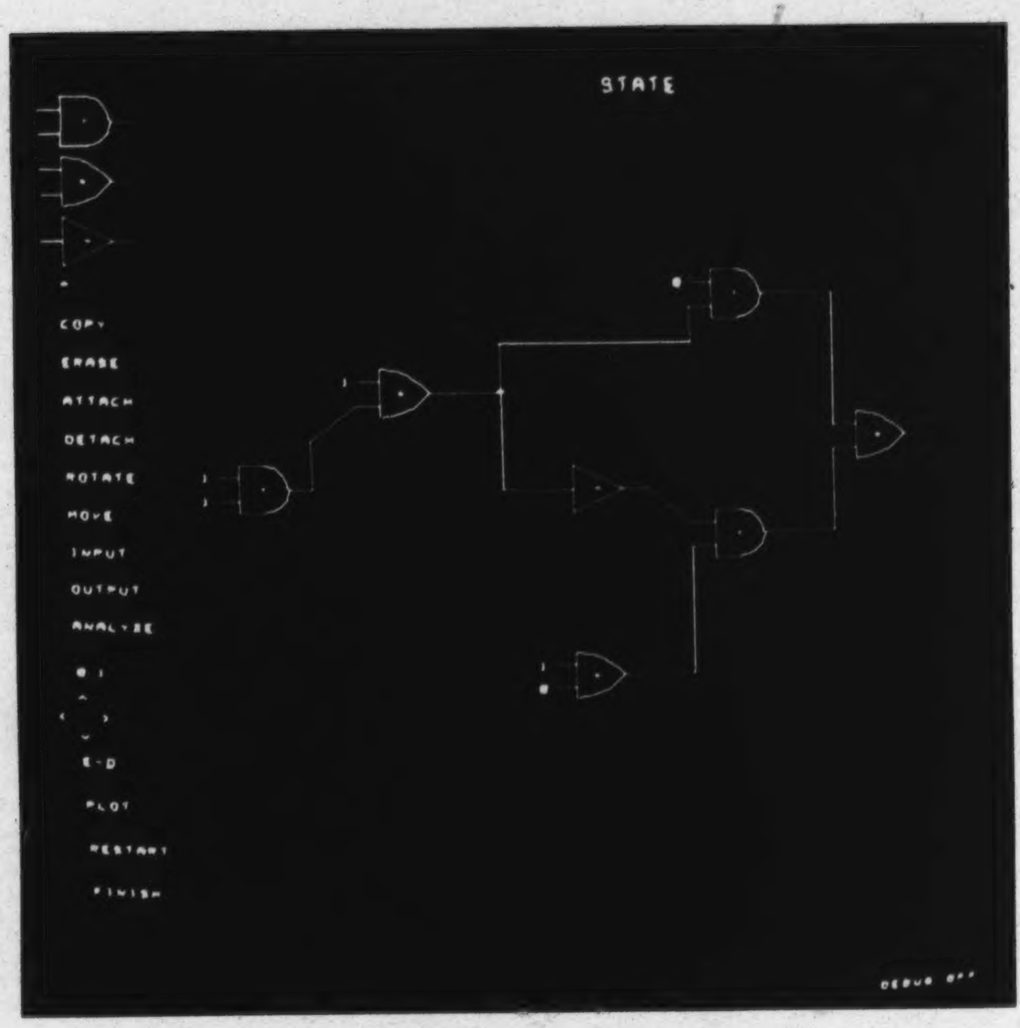

STATE
COPY
ERASE
ATTACH
DETACH
ROTATE
MOVE
INPUT
OUTPUT
ANALYZE
0 1
E-D
PLOT
RESTART
FINISH
DEBUG OFF

Figure 8.2 Johnson's CALD System

of the command.

error_i - is the serial number of the error message to be displayed if the $(i+1)^{th}$ component of the input message is syntactically inadmissable.

module_i - is the name of the procedure (e.g. FORTRAN subroutine name) to be executed if the input command is syntactically correct.

The interpreter has two special modules NULL and ENDJOB. NULL is specified when no processing is required. ENDJOB is used to terminate the program.

For example, in the CALD program the command:

COPY<this component><in this place>,

corresponds to three user actions, each defined in the lexicon:

(1) picking the word COPY with the lightpen (COPY),

(2) selecting one of the prototype gates (GATE),

(3) indicating where on the screen the gate is to be placed (HERE).

The definition of the command for the interpreter preprocessor might be

(COPY 1 NULL) (GATE 2 GATYPE) (HERE 3 COPY) #

where the subroutine GATYPE and COPY are user-supplied modules to determine which type of gate (and, or, not) was picked, and copy the selected gate at the position picked with the lightpen.

The error messages might be:

0 A COMMAND MUST BEGIN WITH ONE OF THE VERBS SPECIFIED ON THE LHS OF THE SCREEN

1 INVALID PROTOTYPE GATE CHOSEN

2 INVALID POINT ON THE SCREEN SELECTED

3 THE COPY COMMAND HAS ONLY 3 COMPONENTS

Error message 0 is displayed if the first component of the input message is syntactically inadmissable. Error message 2, for example, is displayed if the first two components of the input message are COPY and GATE, but the third component is not the action associated with the terminal symbol HERE.

Given that the command has been input correctly, the modules NULL, GATYPE and COPY will be executed in sequence.

8.3 THE PRINCIPAL MODULES FOR THE INTERPRETER

The interpreter consists of a Control Routine plus three subroutines:

(1) Lexical Recognizer.

(2) Analyzer.

(3) Error Handler.

8.3.1 LEXICAL RECOGNIZER

Each of the Boolean expressions ($term_i$) in the lexicon is compiled by a preprocessor into 360 object code. The following ALGOL code illustrates the implementation of the recognizer.

```
procedure lexcheck;
    begin
        integer flag; flag := 0;
        if (term$_1$) then recog (1, flag);
        if (term$_2$) then recog (2, flag);
        if (term$_3$) then recog (3, flag);
        ..............................
        if (term$_n$) then recog (n, flag);
        if flag=0  then error else accept;
    end;
```

The procedure *recog* sets *flag* to one if the terminal symbol recognized may be used by the analyzer.

8.3.2 THE ANALYZER

The CCL is specified as a tabulation of commands. Analysis of the input message is therefore a question of pattern matching.

Formally we may view the task of the analyzer to be: Given a set of patterns

$$\overline{B_i} \qquad i=1,n$$

of varying lengths where

$$\overline{B_i} = b_{i1},b_{i2},\ldots\ldots b_{im} \qquad m\geq 1,$$

accept as input some $\overline{A}$ where

$$\overline{A} = a_1,a_2,\ldots\ldots a_k \qquad k\geq 1.$$

The analyzer is a procedure that decides if

$$\overline{A} = \overline{B_j} \text{ for some } 1\leq j\leq n.$$

If no such j exists, then the analyzer determines the largest $p\geq 0$ and all q such that

$$a_1, a_2, \ldots\ldots a_p = b_{q1}, b_{q2}, \ldots\ldots b_{qp}.$$

Each b_{xy} in B represents a set of triplets ($name_{xy}$, $error_{xy}$, $module_{xy}$). If j exists, then the modules

$$module_{jr} \qquad\qquad r=1,k$$

are queued for execution and executed in sequence. If no such j exists, then for each b_{qp} determined above the related error message $error_{qp}$ is displayed.

In the following discussion we shall use capital letters to represent terminal symbols of the language.

The main problems faced in the construction of the analyzer portion of the interpreter can be demonstrated as follows.

Assume the set of commands to be:

(a) A B C D E #

(b) A F K L #

(c) A B C #

(d) A B #

Consider the following inputs from the terminal:

(1) A BF C D #

(the second element may be either a B or an F)

(2) A F K L C #

(3) A B C K #

(4) A B C #

To illustrate the possibility of an ambiguous input as given in (1) above, consider that B and F are defined:

B ::= $atfk$=3 #

F ::= $atfk$=3∧ats=6 #

If the user presses function key 3 while the status switches have a value of 6, the Boolean expressions associated with B and F both evaluate to TRUE. Thus, both are considered in the analysis of the input message.

The analyzer considers the input from the terminal one element at a time. For input string (1), on receiving the element A and checking the list of commands, all commands are still candidates. On receiving the locally ambiguous BF element, all commands are still candidates. Note that at this time command (d) would be completely decoded. On receiving the third element C, commands (a) and (c) are still candidates, while (b) and (d) are rejected. The fourth element, D, causes the deletion of (c) from the list of candidates. Since there is no E input, the user is given the error message associated with element D in command (a).

For input string (2), we see that after the second element is considered only command (b) remains as a candidate. When the last element C is reached, the error message associated with element L in command (b) is given. For input string (3), after the element C is considered, commands (a) and (c) are still candidates. Since K is not valid in either case, the user is given an error message comprised of two sections, that associated with element C of command (a), and that associated with element C of command (c). For input string (4), after the element C is considered, commands (a) and (c) are still candidates. Since there is no other input element, the analyzer checks to see if any command has been completely decoded. In this instance (c) has been decoded and the object modules named in the triplets for A, B, and C are queued for execution.

8.3.3 THE ERROR HANDLER

Two types of error may occur during the execution of an application program:

(1) syntactic errors due to incorrect user inputs;

(2) execution-time errors found during the execution of the programmer-supplied modules.

For an example of an execution error, consider the <create> command for the Sieve Process, which gives the user the ability to define a new map file. At most 20 such files may be open at any one time. Although the command may be input correctly, when control passes to execution of the <create> request, the user may be given the message

"NO AVAILABLE MAP FILES".

If the first component of an input message is not syntactically admissable for any command, a general error message (associated with error number 0) is displayed on the screen.

An input message may be partially acceptable. When an invalid component is located an error number is obtained from each command that is still a candidate for the target command. The error messages associated with these error numbers are then displayed on the screen.

Messages for both types of error are displayed by the error handler. One block is reserved for display of error messages of either type. If this block is used, the block will be removed from the display file after the input of the next message.

8.4 THE CONTROL ROUTINE

The Control Routine of the Interpreter performs the following steps:

(1) Accept input message from CRT.

(2) Delete error block from display file, if necessary.

(3) Call analyzer and recognizer to decode input.

(4) If input is incorrect, invoke the error handler to select an error message block for display. If input is successfully decoded as a command, queue the appropriate modules for execution and pass control to them in order.

(5) Display resultant display file. Go to step (1) and wait for input message.

8.5 AN EXAMPLE - THE SIEVE PROCESS

For any application the programmer supplies:

(1) The lexicon;

(2) A table of error messages;

(3) A list of all commands, in the form of sequences of triplets;

(4) The program module INITAL and modules required for execution of the commands. INITAL is the module to be used to initialize the tables for the interpreter and the display.

Consider the Sieve Process commands: <display>, <create>, <link>, <window>, <restart>, <stop>, listed in Table 8.2. Each of the command words are displayed within the same block (2) and with unique identifiers. Thus to define the action "lightpen pick of the command word DISPLAY", we add DISPLAY to the lexicon as

DISPLAY ::= $atct=1 \wedge atbn=2 \wedge atid=1$ #

That is, a lightpen pick of something defined within block two and with the identifier = 1. Similarly, we define:

CREATE ::= $atct=1 \wedge atbn=2 \wedge atid=6$ #

CATENATE ::= $atct=1 \wedge atbn=2 \wedge atid=7$ #

WINDOW ::= $atct=1 \wedge atbn=2 \wedge atid=8$ #

RESTART ::= $atct=1 \wedge atbn=2 \wedge atid=9$ #

STOP ::= $atct=1 \wedge atbn=2 \wedge atid=10$ #

The mapnames displayed in the upper left hand corner of the screen (Figure 2.1) have different block numbers, ranging from 40 to 60. The action of picking any one of them is defined:

MAPNAME :: = $atct=1 \wedge atbn \geq 40 \wedge atbn \leq 60$ #

For the <create> command the mapname is to be typed in from the keyboard. This component is defined by:

MAPNAME1 ::= $atct=5 \wedge atn \leq 8$ #

That is, at most 8 characters may be typed in.

The additional <line> for the <link> command may be defined as follows

LINE ::= $atct=4 \wedge atn \leq 25 \wedge atxa<850 \wedge atya<850 \wedge atxb<850 \wedge$
$atyb<850$ #

Table 8.2

Some Sieve Process Commands

<display> → DISPLAY<mapname>

<create> → CREATE<mapname1>

<link> → CATENATE<mapname><line>

<window> → WINDOW<number>

<restart> → RESTART

<stop> → STOP

<mapname> → a lightpen pick of one of the mapnames displayed on the screen.

<line> → a string of vectors input with the lightpen.

<number> → a typed digit from 0 to 8.

<mapname1> → at most eight alphanumerics typed in from the keyboard.

That is, one arbitrary line is drawn with the lightpen, containing at most 24 lines segments (or 25 coordinate pairs); these lines must be entirely within the map area of the CRT.

Input of the <number> for windowing is defined:

DIGIT ::= $atct=5 \wedge atn=1 \wedge atint \leq 8$ #

Only one character may be typed, and, it must be an integer between 0 and 8.

The commands that the user is allowed may then be defined:

(DISPLAY 1 NULL) (MAPNAME 5 DISMAP) #

(CREATE 2 NULL) (MAPNAME1 6 CREATE) #

(CATENATE 1 NULL) (MAPNAME 7 NULL) (LINE 8 CATMAP) #

(WINDOW 10 NULL) (DIGIT 9 WINDOW) #

(RESTART 3 INITAL) #

(STOP 4 ENDJOB) #

The following table of error messages is required:

0	A COMMAND MUST BEGIN WITH ONE OF THE VERBS SPECIFIED ON THE RHS OF THE SCREEN
1	INVALID MAPNAME SELECTED
2	A MAPNAME MUST CONTAIN AT MOST EIGHT CHARACTERS
3	RESTART MUST APPEAR ALONE IN MESSAGE BUFFER
4	STOP MUST APPEAR ALONE IN MESSAGE BUFFER
5	DISPLAY COMMAND HAS ONLY 2 COMPONENTS
6	CREATE COMMAND HAS ONLY 2 COMPONENTS
7	INVALID LINE DATA INPUT
8	CATENATE COMMAND HAS ONLY 3 COMPONENTS
9	WINDOW COMMAND HAS ONLY 2 COMPONENTS
10	WINDOW NUMBERS RANGE FROM 0 TO 8

The modules DISMAP, CREATE, CATMAP, WINDOW and INITAL are user supplied. DISMAP displays one map on the screen using the current scaling and translation factors. CREATE creates an empty map file and gives it the label typed in. CATMAP catenates to a map file the new line drawn by the user. WINDOW changes the scale of presentation and the centre of view and redisplays at the new scale all maps currently displayed. INITAL initializes the display and the tables for the interpreter.

CHAPTER IX

IMPLEMENTATION AND DISCUSSION

9.1 INTRODUCTION

The man-machine interface discussed in Chapter 8 has been implemented by the author for use at the University of Alberta, and used both by the author and others in programming graphics applications. The Appendix contains a description of the subroutines and global variables for use by the application programmer. The main features of the system are presented in Section 9.2.

As a man-machine interface, the interpreter has a number of virtues. Its use makes programming and debugging easier because of modularization. It has the advantage common to any essentially table-driven interpreter, that the CCL may easily be modified, and evolve from an initial form to one more convenient for the user.

The present implementation of the interpreter is in two sections. The GRID terminal has a supervisor which accepts user actions and transmits to the 360 the contents of a message buffer when the user presses the SEND key. The main part of the interpreter (lexical analyzer, recognizer and error handler) is resident in the 360.

The system satisfies most, but not all, of the objectives set out in Section 7.4. The chief shortcoming is that syntax errors are not discovered until the input is transmitted to the 360. In Sections 9.3, 9.4 and 9.5, the author discusses an improved system which would satisfy all these objectives.

In Sections 9.6 and 9.7, this system is compared to one proposed by Morrison (1967).

9.2 INTERACTION WITH THE INTERPRETER

Three sets of data are required by the interpreter:

(1) the lexicon,

(2) the console commands as a set of triplets (described in Section 8.2.2),

(3) the error messages.

The procedures TERMS, COMAND, and ERRORS are preprocessors used to convert these data sets from the form supplied by the programmer to the form required by the interpreter. The programmer can therefore check, to some extent, the specification of the CCL before using it in an application program. The procedures RECOGT, RECOGC and RECOGE are used to inform the interpreter of the address and size of each of the tables.

By separating the two functions of preprocessing input tables and linking the processed tables to the interpreter, the programmer is offered additional flexibility. Preprocessed tables may be saved, for example on disc files, from session to session. At any time during the execution of the application program, the tables used by the interpreter may be modified.

Three areas of labelled common are used by the FORTRAN programmer:

```
COMMON/ATVALS/ATCT,ATBN,.....
```

where ATCT,ATBN,....., the attention variables of

Table 8.1, are each integer variables.

COMMON/APARMS/ACHRMS,AXYMAX,ACHAR,AX,AY

where ACHRMS and AXYMAX are integer variables;

ACHAR,AX,AY, are integer vectors of lengths ACHRMS, AXYMAX and AXYMAX respectively, used as indicated in Table 8.1.

COMMON/SETING/BUNCH,EOM,ERROR,DISCOM

where BUNCH,EOM,ERROR,DISCOM, are logical variables used as described in the Appendix.

The first two common areas make available the attention variables that record the input actions. In the third set of common variables, the variable DISCOM is a flag to indicate whether the current input message from the CRT is to be saved on a disc file for later analysis. This facility provides a useful debugging aid, and the programmer may also, by seeing what commands are improperly used, improve the command language to make it more natural for the user.

9.3 DISCUSSION

Let us consider whether the user-program interface satisfies the design criteria established in Section 7.4. The first and third criteria are that there should be a standard interface between the two machines and that the user need program for only one machine in one language. These two objectives are satisfied. The interface between the 360 and the GRID is at the level of a "Report Block". The interface does not change with any change of application program. At the same time, the programmer need only program in one language.

He must supply the support routines, to be executed if a command is input, but these may be in the language of his choice. The author feels that the input format for lexical elements, error messages or commands is not sufficiently complex to be deemed a different language from either FORTRAN (or ALGOL). These modules may be written as if all programs are to be executed in the 360.

The second criterion, is that there should be an immediate reaction to each input and syntactic errors should be reported as they occur. While there is an immediate reaction to each input, syntax errors are discovered only when the message is transmitted to the 360. This immediate reaction takes the form of:

(1) Displaying alphanumeric input on the screen as it is typed.

(2) Displaying a dot at each point in a strings of points being defined.

(3) Displaying for each line, in a sequence of vectors, the line as it is described.

(4) Displaying the number of each function key as it is picked.

(5) Displaying an arrow pointing to any displayed element picked with the lightpen.

The fourth objective is that the system be simple to understand and use. The author believes that this objective is satisfied.

Let us consider how the second objective might be satisfied. The configuration for which this software has been implemented comprises a programmable terminal coupled to a large time-shared system. In our view, this is a configuration well suited for interactive graphics. Ideally, for such a system, the main machine would interpretively execute complete commands input by the user, with the programmable terminal handling all interaction during input of a command.

At present the GRID supervisor allows:

(1) Displayed items to be picked with the lightpen.

(2) Function key inputs.

(3) Input from the keyboard of alphanumeric strings.

(4) A string of points or lines to be input, using the lightpen and function keys, in blank areas of the screen.

For each action a visible response is given, and appropriate data added to the message buffer to be transmitted to the 360.

To meet our requirements for the terminal to handle immediate interaction with the user, syntax analysis must be done for each input action as it occurs. The lexical recognizer, the syntax analyzer and the error handler, (Section 8.3) should therefore be in the terminal. For this to be possible, modifications are required to the lexical recognizer and the syntax analyzer.

To hold these modules in the GRID terminal appears impractical because of core limitations. Space required for error messages is a key factor. The GRID processor has 12K words of core memory, 8K words being reserved for the display file and the remaining 4K used by the supervisor. Possible ways of overcoming this are:

(1) Compaction of error messages.

(2) Use of a disc on the small machine.

Wagner (1973) has pointed out that storage requirements for error messages may be minimized by storing, as character strings, common components of error messages (phrases) and substituting pointers to the phrases where they occur in the error messages. This technique was tested for the error message table given in Section 8.5 with the following results: Stored entirely as characters, the messages require 232 words. Using Wagner's technique 186 words are required. It is clear that for an entire program, core requirements for error messages would still be excessive. Use of a small disc or cassette tape for storage of these messages is warranted.

9.4 MODIFIED ANALYSIS OF INPUT

Ideally, each user input at the terminal should either be accepted with a visible response, or rejected and an error message issued. For each user action, there must be an immediate analysis to see if it is acceptable as part of a command. Figure 9.1[*] illustrates the organization of such a system. If on analysis the input is inadmissable, an error message is displayed and program control reverts to that

---

[*] This disgram is to be used for a comparison against a system proposed by Morrison (1967) - see Figure 9.2.

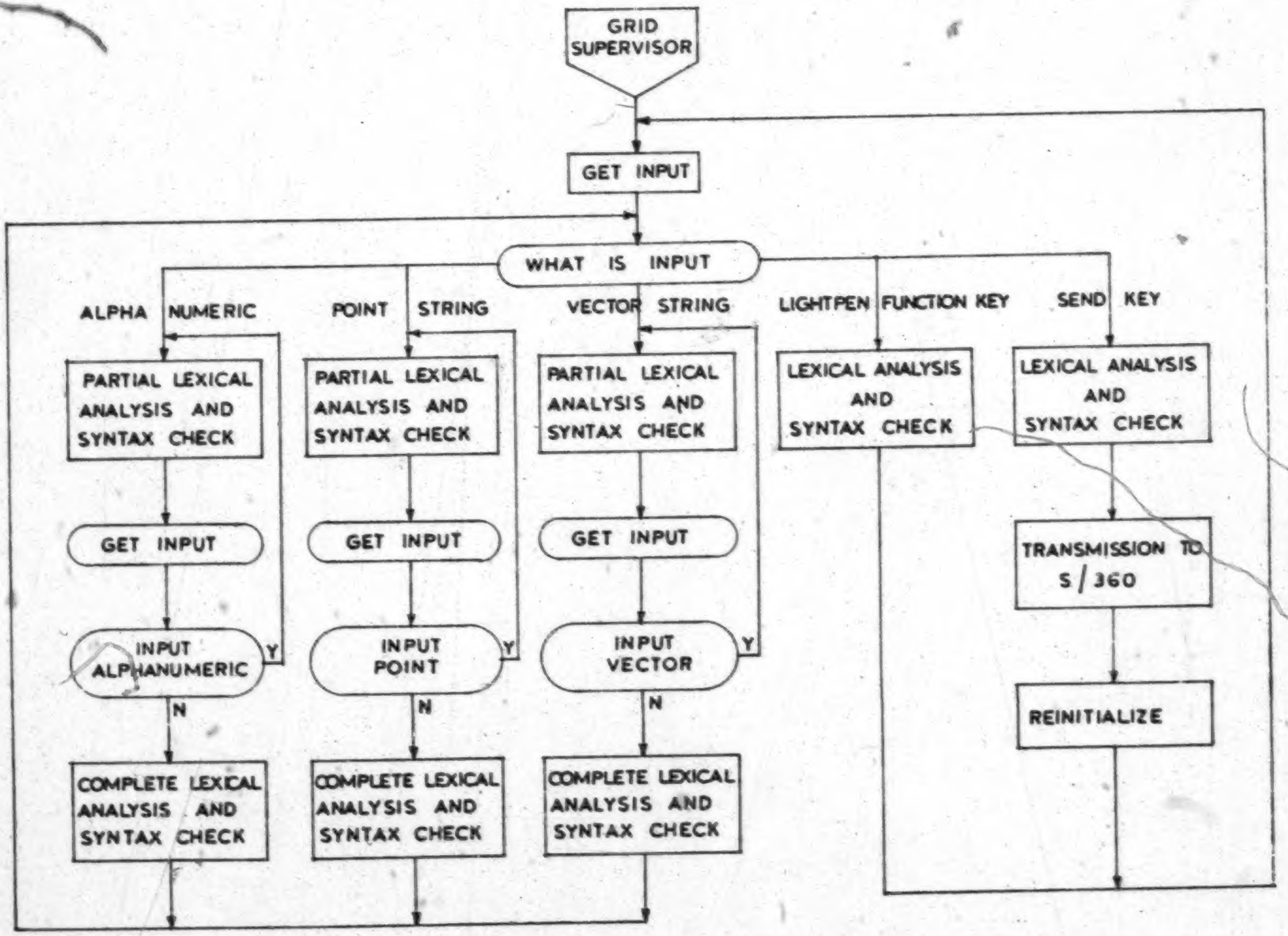

GRID SUPERVISOR
GET INPUT
WHAT IS INPUT
ALPHA NUMERIC
POINT STRING
VECTOR STRING
LIGHTPEN FUNCTION KEY
SEND KEY
PARTIAL LEXICAL ANALYSIS AND SYNTAX CHECK
PARTIAL LEXICAL ANALYSIS AND SYNTAX CHECK
PARTIAL LEXICAL ANALYSIS AND SYNTAX CHECK
LEXICAL ANALYSIS AND SYNTAX CHECK
LEXICAL ANALYSIS AND SYNTAX CHECK
GET INPUT
GET INPUT
GET INPUT
TRANSMISSION TO S/360
INPUT ALPHANUMERIC
Y
N
INPUT POINT
Y
N
INPUT VECTOR
Y
N
REINITIALIZE
COMPLETE LEXICAL ANALYSIS AND SYNTAX CHECK
COMPLETE LEXICAL ANALYSIS AND SYNTAX CHECK
COMPLETE LEXICAL ANALYSIS AND SYNTAX CHECK

Figure 9.1 Logical Structure of a new GRID Supervisor

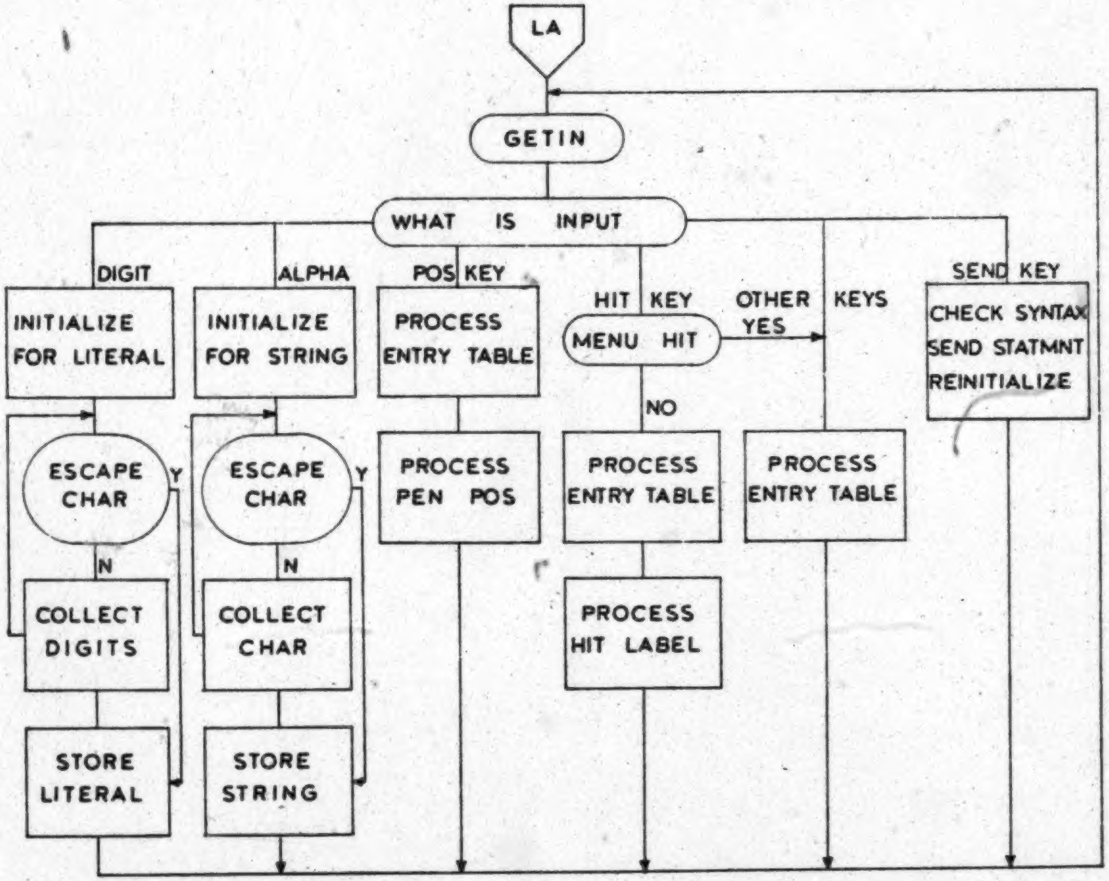

LA
GETIN
WHAT IS INPUT
DIGIT
ALPHA
POS KEY
HIT KEY
OTHER KEYS
SEND KEY
INITIALIZE FOR LITERAL
INITIALIZE FOR STRING
PROCESS ENTRY TABLE
MENU HIT
YES
NO
CHECK SYNTAX
SEND STATMNT
REINITIALIZE
ESCAPE CHAR
Y
N
ESCAPE CHAR
Y
N
PROCESS PEN POS
PROCESS ENTRY TABLE
PROCESS ENTRY TABLE
COLLECT DIGITS
COLLECT CHAR
PROCESS HIT LABEL
STORE LITERAL
STORE STRING

Figure 9.2 Logical Structure of Morrison's Lexical Analyzer

portion of the program (the GET INPUT box) which accepted the last input.

To provide for immediate acceptance or rejection, syntax analysis must be done on incomplete commands. There is also a further requirement that partial checking be done at the lexical level. Of the five component types defined for use with the GRID (Section 8.2), three (Point, Vector and Character) are strings of inputs rather than single inputs.

Each user action in the string of inputs for any one of these component types must be checked individually. That is, we must also perform lexical checking on a "partial" entry.

If an input of a different component type is given (Morrison denotes this as an "escape character" in Figure 9.2), the "partial" entry must be checked as a "complete" entry and either accepted or rejected. If the "complete" entry is acceptable, then the "escape character" can be checked.

Consider, for example, the lexical entry:

MAPNAME ::= $atct=5 \wedge atn \leq 8 \wedge atn \geq 3$ #

which defines the action MAPNAME to be the input of a string of not less than 3 and not more than 8 alphanumeric characters. Each character input must be either accepted or rejected. If, on analysis of the first character input, MAPNAME is accepted as a possible next element in a command[*], the character would

[*] The example shows also the interaction between syntactic and lexical analysis. If MAPNAME appears only in the context

(CATENATE 2 NULL) (MAPNAME 3 NULL) (LINE 4 CATMAP) #

then MAPNAME is accepted as a possible next element in a command only if the entry CATENATE preceded it and at least three alphanumeric characters are input.

be accepted. If, after only two characters are input, an action of a different component type is input, this "escape character" would not be accepted, unless the "complete" character string is recognized as some other entry in the lexicon that is an acceptable next element in a command. Instead, an error message would be displayed on the screen. However, the user can then type the third character of the string.

This partial checking may be accomplished as follows. Of the attention variables associated with the three component types Point, Vector and Character, only one, *atn*, is concerned with the number of elements in the string. "Partial" checking will not involve analysis of any conditions of the form $atn>X$ or $atn\geq X$. If a condition of the form $atn=X$ is given, the preprocessor will expand it to $atn=X \wedge atn<X+1$ and the condition $atn=X$ will not be tested during partial analysis. When the "complete" input is available, all conditions will be tested before acceptance of the "escape character". The attention variables *atreal* and *atint*, variables relating to the input of floating-point numbers and integers from the keyboard, may be handled in a similar manner.

Since the analyzer described in Section 8.3.2 accepts inputs one at a time, it does not require serious modification. Changes to the system would include modifications to enable the analyzer to:

(1) Handle inputs that have undergone partial analysis.

(2) Decide whether a complete command has been input

and therefore to accept or reject a request to "send" the buffer to the 360.

By imbedding the modified lexical recognizer, syntax analyzer and the error handler in the GRID supervisor, each input, when it occurs, may be accepted or rejected. If an input is rejected an explanatory error messages may be displayed.

9.5 EXAMPLES

Let us suppose that the modified lexical recognizer, syntax analyzer and error handler were to reside in the terminal. That the system would have great power for error checking can be seen from a number of examples.

Any input actions, to be acceptable, must be first recognized as a member of the lexicon. Then, the action must be acceptable with those actions which have preceded it.

For example, consider the <link> command of the Sieve Process. The command has three components:

(CATENATE .....) (MAPNAME .....) (LINE .....) #

If the user picks the word CATENATE, he must follow by picking a MAPNAME from the top left corner of the screen. If he attempts some other action before picking the map name, then the action is rejected with the message

PICK A MAP NAME FROM THOSE DISPLAYED ON THE SCREEN*.

---

* The messages specified in Section 8.5 were intended for the system with syntax analysis in the 360. If the syntax analyzer were in the terminal, then one would design error messages which would guide the user.

LINE is defined in the lexicon as:

LINE ::= $\mathit{atct}=4\wedge\mathit{atn}\leq 25\wedge((\mathit{atxa}<850\wedge\mathit{atya}<850)\wedge$
$(\mathit{atxb}<850\wedge\mathit{atyb}<850))$ #

that is, a sequence of at most twenty four vectors each of them within the specified area of the screen. If an attempt is made to draw outside this area the vector is not accepted, with the message

INVALID DATA POINT.

He can, however, continue by simply ensuring that the vector is within the designated area.

As a second example suppose that in a circuit-design system the value of the resistance must be input. The value is to be typed after picking the word "RESISTANCE" and the appropriate resistor. Let us suppose that resistances are required to be between 100 and 1000 ohms. The action can be defined

RESIST ::= $\mathit{atct}=5\wedge(\mathit{atint}\geq 100\wedge\mathit{atint}\leq 1000)$ #

As a final example, suppose that the input to a question must be either T or F. The lexical entry would be

TF ::= $\mathit{atct}=5\wedge\mathit{atn}=1\wedge(\mathit{atcode}=\text{'T'}\vee\mathit{atcode}=\text{'F'})$ #

Software design for graphics terminals often makes use of "selective enabling" (for example, see Beckermeyer (1970)). To assist the user, entities may be disabled (that is, made "invisible" to the lightpen) for those periods during which they should not be picked.

Selective enabling, not merely for lightpen pick but for any type of action, could be achieved simply by adding or deleting entries in the lexicon.

9.6 MORRISON'S LANGUAGE PROCESSOR

An interface, designed by Morrison (1967) for use with an "intelligent terminal", shows similarities with the system which we have described. Morrison notes that, while it is, in principle, possible to let the input stream directly invoke the necessary procedures, it is convenient to interpose a lexical analyzer between the graphic input devices and the procedures themselves. The duties of this lexical analyzer include isolating identifiers, literals, and the operators and delimiters of the graphic language. More important, however, it handles the switching of displayed menus as the user makes his selections with the lightpen, and certain graphic manipulations (e.g. lightpen tracking, picture moving, scaling, etc.). It also does some primitive syntax checking.

This language processor is modularized into two subsystems. "The first subsystem, the lexical analyzer, deals with the input language from the graphic console to a very shallow level. It is concerned only with the words or vocabulary of the language with little concern for the structure (syntax) and meaning (semantics) of the language. It recognizes inputs from the function keys, typewriter, and lightpen, sorts them out and, where necessary, concatenates them into identifiers, real and integer literals, and the terminal symbols of the language.

The second subsystem, the statement subroutines, is concerned with the syntax and semantics of the graphic language. A statement subroutine exists for each statement in the language. It performs the actions indicated by the semantics of the statement. Many of the statement subroutines perform similar actions (i.e. manipulate or add to the data base, and add items to the display file)." In general, the "statement subroutines" are developed by the programmer for the particular application under consideration.

Figure 9.2 illustrates Morrison's lexical analyzer. In summary, the analyzer may be divided into two parts, one identifying the component type of the user actions, the other carrying out primitive syntax checking.

Syntax checking is done to the extent that:

(1) Each element in the command must be of the correct component type.

(2) Each compound component must have the correct number of partial elements.

(3) The correct number of components for the command must be input.

9.7 DISCUSSION

Of the objectives for our own system in Section 7.4, Morrison's interface appears to satisfy only objective (1), though objective (2) is satisfied to some degree. It is clear that not all syntax errors are discovered. If the input is of the proper component type and has the correct number of parts then it is accepted. The analyzer is table-driven and Morrison notes that a sub-language is proposed for use in defining the table. Thus, we are unable to say whether objective (3) is attained. That the facility is easily understood and used can only be verified by "hands on" experience.

The GRID supervisor accomplishes part (1) of Morrison's analyzer. Each input is categorized as to component type and relevant data included in the message buffer for transmission to the 360. Part (2) of Morrison's analyzer performs analysis of input at the level of component type. This analysis, in more detail, is performed in the author's system in the main machine. However, as is seen in Sections 9.3, 9.4 and 9.5, this analysis could be transferred to the terminal, given another 4K of storage or a small disc. Detailed analysis, in Morrison's system, is done by user-supplied modules in the main machine.

The author believes that he has shown his system to be very much more powerful than that described by Morrison.

9.8 SUMMARY

The author has developed a man-machine software interface for use with interactive graphics programs. The interface is based on a formalism enabling the programmer to describe user actions as terminal symbols in a Console Command Language. A table-driven interpreter for such a language has been developed.

The system has several major benefits, the most important being that:

(1) Design by the application programmer of a man-machine interface with powerful error-checking has become a straightforward task.

(2) With the CCL explicitly specified in table form, modification of the language is very simple.

(3) The programs to be provided by the applications programmer can be supplied as separate modules.

This type of graphics software seems ideally suited for a configuration comprising a programmable terminal coupled to a large time-shared system. The supervisor in the terminal would handle interaction with an immediate response to user actions. The application programmer, however, need program in only one language for what appears to be one machine.

The overall design also seems very suitable for a stand-alone system making use of a small to medium sized processor with substantial disc storage. Suppose that the routines resident in core comprised the lexical recognizer, syntax analyzer, error handler, and the control routine (Section 8.4), while all other routines were held on disc. In our present system, the control routine queues modules for execution. It would be relatively simple for the control routine to bring the required modules into core for execution in sequence.

CHAPTER X

CONCLUSIONS

At the beginning of this thesis, we stated that complex computer systems to be used for planning are a long way off with many problems needing to be solved. In this thesis, a number of these problems have been considered.

The implementation of the Sieve Process has shown that the map storage and retrieval functions required in planning can be successfully and conveniently handled by the computer. For such a system to be used efficiently and economically on a large scale, great care must be taken in choosing methods for representing and structuring map data.

The method chosen for data representation can be particularly significant. For example, the most frequently used function in an interactive map retrieval system is that used to display data. The processing time required to apply this function is generally proportional to the number of data elements, and it is therefore important that the data be economically encoded. Three commonly used methods of data representation were studied, and the results obtained differ substantially from those published elsewhere. It is believed that this can be explained, at least in part, by the fact that the maps considered by the author were significantly different than those considered elsewhere in the literature. It has been shown, however, that the relative merits of the methods considered vary greatly with both the scale and the error allowed in encoded data.

Another problem considered was that of defining a data base of information on resources available to the planner. Such a data base is required if we are to transfer to the computer the task of evaluating, other than superficially, alternative land-sites. The author has specified what he considers to be the main design criteria for a data base to be used in an interactive environment. In Chapter 6, a description of the structure of such a data base is presented. A major feature of the data base is that it may contain data common to many planning projects, yet at the same time be readily modifiable so that it may be of particular use for one problem.

A major obstacle to wider use of interactive graphics is the time required to develop a suitable man-machine software interface. The author has designed and implemented a new graphics interface. A formalism has been developed for describing user actions at the CRT as terminal symbols in a Console Command Language. The formalism is the basis for a table-driven interpreter for use with any Console Command Language. Use of the interface simplifies greatly the work needed to provide for graphical communication in an interactive graphics program. At the same time, the other programming requirements are modularized enabling the programmer to implement the commands one at a time without interference from other sections of the program.

An improved version of the interface, designed to reside in the terminal rather than in the main machine, has been considered. A major advantage of this system is that it would detect a very wide range of errors by the user.

The ultimate goal should be to provide what one might ambitiously call a "total planning system". The development of any very complex application requires solution of many problems, not all of which can be covered in this thesis.

We have considered the relative merits of map encoding methods with respect to storage economy. One should also consider algorithms for map processing functions for each type of data representation. Intensive study has been made of numerical algorithms. Algorithms for the manipulation of map data, a field that may be of equal importance, deserve more attention.

The data structure presented in Chapter 6 has enabled us to transfer to the machine some of the problems of evaluating alternative land-sites. It remains to be seen, however, whether the structure is suitable for use with other techniques, such as linear programming, which may be employed to determine an optimal solution to a given problem. The design of better graphics support software is itself a major task with many aspects.

These problems vary widely in nature. Better solutions to each of them must be sought if there is to be general use of the computer other than as an occasional tool in the solution of planning problems. The author believes, however, that the work described in this thesis shows significant progress on some of the more significant problems.

BIBLIOGRAPHY

Aguilar, R.J., Optimality in Facility Location, Presented at the Joint Meeting of Canadian Institute of Architects and University of Alberta Extension Division, Edmonton, 1969.

Amidon, E.L., and Akin, G.S., "Algorithmic Selection of the Best Method for Compressing Map Data Strings", C.A.C.M., Vol. 14, No. 12, pp. 769-774, 1971.

Amsterdam, R., "Implementation of the GIST Geographic Base File", Proc. 26th National Conf., ACM, pp. 315-324, 1971.

Beckermeyer, R.L., "Interactive Graphic Consoles - Environment and Software", Proc. Fall Joint Computer Conf., AFIPS, Vol. 37, pp. 315-323, 1970.

Boullier, P., Gros, J., Jancene, P., Lemaire, A., Prusker, F., and Saltel, E., "METAVISU, A General Purpose Graphic System", draft of paper presented at IFIP meeting on Graphics Programming Languages at Vancouver, 1972.

Boyle, A.R., "Computer Aided Map Compilations", Pre-Seminar Digest 2nd Man-Computer Communication Seminar Interactive Graphics, NRC, Ottawa, pp. 46-52, 1971.

Chen, F.C., and Dougherty, R.L., "A System for Implementing Interactive Applications", IBM Systems Journal, Vol. 7, No. 3 & 4, pp. 257-270, 1968.

Christensen, C., and Pinson, E.N., "Multi-Function Graphics for a Large Computer System", Proc. Fall Joint Computer Conf., AFIPS, Vol. 31, pp. 697-711, 1967.

Cook, B.G., "A Computer Representation of Plane Region Boundaries", Australian Computer Journal, Vol. 1, No. 1, pp. 44-50, 1967.

Cotton, I.W., and Greatorex, F.S., "Data Structures for Remote Computer Graphics", Proc. Fall Joint Computer Conf., AFIPS, Vol. 33, pp. 533-544, 1968.

Davis, M.R., and Ellis, T.O., "The Rand Tablet: A Man-Machine Graphical Communication Device", Proc. Fall Joint Computer Conf., AFIPS, Vol. 26, pp. 325-331, 1964.

Deecker, G.F.P., Interactive Graphics and a Planning Problem, M.Sc. Thesis, Dept. of Computing Science, University of Alberta, 1970.

Deecker, G.F.P., and Penny, J.P., "On Interactive Map Storage and Retrieval", INFOR, Vol. 14, No. 1, pp. 62-74, 1972.

Dror, Y., Introduction to a General Theory of Planning, Institute of Social Studies, The Hague, 1961.

Earley, J., "Toward an Understanding of Data Structures", C.A.C.M., Vol. 14, No. 10, pp. 617-627, 1971.

Enerson, R.C., A Data Structure Approach to Interactive Graphics Software, M.Sc. Thesis, Department of Computing Science, University of Alberta, 1972.

Freeman, H., "On the Encoding of Arbitrary Geometric Configurations", IRE Transactions on Electronic Computers, Vol. EC-10, No. 2, pp. 260-268, 1961.

Freeman, H., "Techniques for the Digital Computer Analysis of Chain-Encoded Arbitrary Plane Curves", Proc. National Electronics Conf., Vol. 17, pp. 421-432, 1961.

Freeman, H., and Glass, J.M., "On the Quantization of Line-Drawing Data", IEEE Transactions, Vol. SSC-5, No. 1, pp. 70-79, 1969.

Freeman, H., "Boundary Encoding and Processing", Pictorial Processing and Psychopictorics, Academic Press, pp. 241-266, 1970.

Gray, J.C., "Compound Data Structures for Computer Aided Design; a Survey", Proc. 22nd National Conf., ACM, pp. 355-365, 1967.

Hamilton, J.A., A Survey of Data Structures for Interactive Graphics, Memo RM-6145-ARPA, Rand Corporation, 1970.

Huen, W.H., A Graphical Display Subroutine Package, M.Sc. Thesis, Dept. of Computing Science, University of Alberta, Edmonton, Alberta, 1969.

Jackson, W.C., Computer Graphics for the Applications Programmer, Dept. of Computing Services, University of Alberta, 1972.

Johnson, B.V., Computer Graphics in Logic Circuit Design, M.Sc. Thesis, Dept. of Computing Science, University of Alberta, Edmonton, Alberta, 1969.

Joyce, J.D., and Cianciolo, M.J., "Reactive Displays: Improving Man-Machine Graphical Communication", Proc. Fall Joint Computer Conf., AFIPS, Vol. 31, pp. 713-721, 1967.

Machover, C., "The Intelligent Terminal", in Pertinent Concepts in Computer Graphics, pp. 179-199, 1969.

Machover, C., "CRT Graphic Terminals", Australian Computer Journal, Vol. 2, No. 3, pp. 117-135, 1970.

Maclean, M.A., "Designing a Language for Interactive Control Programs", Pre-Seminar Digest 2nd Man-Computer Communication Seminar Interactive Graphics, NRC, Ottawa, pp. 30-39, 1971.

Mezei, L., "SPARTA, a Procedure Oriented Programming Language for the Manipulation of Arbitrary Line Drawings", IFIP Conf. Proc., pp. C96-C102, 1968.

Morrison, R.A., "Graphic Language Translation with a Language Independent Processor", Proc. Fall Joint Computer Conf., Vol. 31, pp. 723-731, 1967.

Newman, W.M., "A System for Interactive Graphical Programming", Proc. Spring Joint Computer Conf., AFIPS, Vol. 32, pp. 47-54, 1968.

Newman, W.M., "A High-Level Programming System for a Remote Time-Shared Graphics Terminal", Pertinent Concepts in Computer Graphics, pp. 200-223, 1969.

Newman, W.M., and Sproull, R.F., Principles of Interactive Computer Graphics, McGraw Hill, 1973.

Ninke, W.H., "A Satellite Display Console System for a Multi-Access Central Computer", Proc. IFIP Congress 1968, pp. 962-969, 1968.

Parker, D.B., Solving Design Problems in Graphical Dialogue, Programming Technical Report TER-01, C.D.C., 1966.

Penny, J.P., Deecker, G.F.P., and Ng, N., "On General Purpose Software for Interactive Graphics", Proc. of the Fifth Australian Computer Conference, 1972.

Peucker, T.K., Computer Cartography: A Working Bibliography, Discussion Paper No. 12, Dept. of Geography, University of Toronto, 1972.

Peucker, T.K., Computer Cartography, Commission on College Geography Resource Paper No. 17, Association of American Geographers, Washington, 1972.

Pfaltz, J.L., and Rosenfeld, A., "Computer Representation of Planar Regions by Their Skeletons", C.A.C.M., Vol. 10, No. 2, pp. 119-125, 1967.

Rosenfeld, A., and Pfaltz, J.L., "Sequential Operations in Digital Processing", J.A.C.M., Vol. 13, No. 4, pp. 471-494, 1966.

Schumacker, B., "URBAN-COGO - A Geographic-Based Land Information System", Proc. Fall Joint Computer Conf., AFIPS, Vol. 39, pp. 619-630, 1971.

SIGGRAPH, "Report of the Greater Boston Chapter of SIGGRAPH", Computer Graphics, Quarterly Report of SIGGRAPH-ACM, Winter 1969.

Tomlinson, R.F., An Introduction to the Geo-Information System of the Canada Land Inventory, Ministry of Forestry and Rural Development, Ottawa, 1967.

Tomlinson, R.F., Ed., Geographical Data Handling, International Geographical Union Commission on Geographical Data Sensing and Processing, 1972.

University of Alberta, The University of Alberta North Garneau Development: Interim Report: Planning, Bittorf Pinckston Planning Consultants, Edmonton, 1968.

Van Dam, A., and Evans, D., "A Compact Data Structure for Storing, Retrieving and Manipulating Line Drawings", Proc. Spring Joint Computer Conf., AFIPS, Vol. 30, pp. 601-610, 1967.

Wagner, R.A., "Common Phrases and Minimum-Space Text Storage", C.A.C.M., Vol. 16, No. 3, pp. 148-152, 1973.

APPENDIX

A GRAPHICS MAN-MACHINE SOFTWARE INTERFACE

A.1 INTRODUCTORY DESCRIPTION

A package of assembler language routines callable from a FORTRAN program is available for defining and implementing, by means of a table-driven interpreter, a Console Command Language for communication between the FORTRAN program and a user at the CRT. The routines are to be used under the Michigan Terminal System (MTS). The routines with minor modifications, may be used under OS/MVT. The modifications include programming system subroutines available in MTS.

For any application the user must supply in tabular form a specification of:

(1) user actions;

(2) error messages;

(3) allowable commands, and a mapping of these commands onto subroutine names and error messages.

Three preprocessors, TERMS, ERRORS and COMAND, are used to process this input into a form acceptable to the interpreter. The resulting data is stored in user-specified memory locations. Three modules, RECOGT, RECOGE, and RECOGC, are available to inform the interpreter of the address and size of tables containing the desired terminal symbols, error messages, and commands for a particular session at the CRT.

These six routines, together with three labelled common areas, constitute the means for programmer interaction with and control of the interpreter.

Jackson (1972) notes that to use GRIDSUB, the user's "main program" is the subroutine GRAFIC. The interpreter developed by the author takes the place of this GRAFIC subroutine. Instead, the user supplies the subroutine INITAL to initialize the display file and the tables for the interpreter.

A.2 DETAILED DESCRIPTION OF THE PROCEDURES

A.2.1 INPUT FORMAT

Each of the preprocessor routines accepts input from MTS defined line files. Initially these may be card input files.

The lexical entries, describing the primitives of the CCL, are input to the procedure TERMS. Their description is format free except for the restriction that each <term> must start on a new input line. The syntax for any <term> is given in Table A.1. There may be at most seven levels of nested parenthesis. The set of terms is delimited by #.

Error messages are input to the procedure ERRORS using the FORTRAN format (I4,19A4). The set of error messages is delimited by #.

The triplets (described in Section 8.2.2) representing the command components are input to the procedure COMAND. Their description is format free with the exception that the definition of any one triplet must not be spread over two input records. The set of triplets for any command is delimited by #. The set of commands is also delimited by #.

Table A.1

Syntax for Lexical Entries

<term> → <item> ::= <Boolean expression> #

<Boolean expression> → <term>|<Boolean expression>∨<term>

<term> → <primary>|<term>∧<primary>

<primary> → <attention variable><rel op><integer>|
atreal <rel op><real number>|
(<Boolean expression>)

<rel op> → >|<|≥|≤|=

<integer> → <unsigned number>|-<unsigned number>

<unsigned number> → <digit>|<unsigned number><digit>

<real number> → <integer>.|<integer>.<unsigned number>

<item> → $\{\langle \mathrm{EBCDIC} \rangle\}_1^8$

<EBCDIC> → A|B|C|D|E|F|G|H|I|J|K|L|M|N|O|P|Q|R|
S|T|U|V|W|X|Y|Z|<digit>|_|$|¢|+|?|*

<digit> → 0|1|2|3|4|5|6|7|8|9

<attention variable> → *atct*|*atbn*|*attype*|*atxp* ..... the
attention variables listed in Table 8.1

A.2.2 PREPROCESSOR ROUTINES

(1) TERMS (CODE,TRMLST,FILEN,TSIZE,MAXC,MAXT,&ERROR)

Parameters

CODE: Mode Integer array; dimension MAXC
TRMLST: Mode Integer array; dimension MAXT
FILEN: Mode Integer
TSIZE: Mode Integer
MAXC: Mode Integer; the dimension of CODE
MAXT: Mode Integer; the dimension of TRMLST
&ERROR: ERROR is a statement number.

Function

The lexical entries defining user actions as Boolean expressions are read from the logical device FILEN. The expressions are compiled to IBM/360 machine instructions in the array CODE. The code is equivalent to the Algol Procedure described in Section 8.3.1. On exit, the array TRMLST contains (in A4 format) the names of the entries. TSIZE is set to the number of entries processed.

If the execution of the subroutine terminates abnormally, control is handed to the statement specified by ERROR.

Possible Errors

(1) End of file on logical device FILEN.
(2) CODE not dimensioned to be large enough.
(3) TRMLST not dimensioned to be large enough.
(4) Syntax error is description of user action.

(2) ERRORS (ERTABL,FILEN,ESIZE,MAXE,&ERROR)

Parameters

ERTABL: Mode Integer array; dimension MAXE

FILEN: Mode Integer

ESIZE: Mode Integer

MAXE: Mode Integer; the dimension of ERTABL

&ERROR: ERROR is a statement number.

Function

To preprocess the error messages on logical device FILEN to the form required by the interpreter.

On exit, ERTABL contains the formatted error messages and ESIZE indicates how much of the array ERTABL has been used.

If the subroutine terminates abnormally, control is handed to the statement specified by ERROR.

Possible Errors

(1) End of file on logical device FILEN.

(2) ERTABL not dimensioned to be large enough.

(3) COMAND (COMMS,FILEN,CSIZE,TRMLST,TSIZE,MAXC,&ERROR)

Parameters

COMMS: Mode Integer array; dimension MAXC

FILEN: Mode Integer

CSIZE: Mode Integer

TRMLST: Mode Integer array

TSIZE: Mode Integer

MAXC: Mode Integer; the dimension of COMMS

&ERROR: ERROR is a statement number.

Function

To preprocess the commands to the form required by the interpreter. Commands are read from the logical device FILEN. Each command component is a triplet as described in Section 8.2. The data structure to be used by the interpreter is built in the array space COMMS. TRMLST is an array created by the procedure TERMS. TSIZE indicates the number of terminals defined in TRMLST. On exit, CSIZE indicates the amount of space used in the array COMMS.

If execution terminates abnormally, control is handed to the statement specified by ERROR.

Possible Errors

(1) COMMS not dimensioned to be large enough.
(2) End of file on logical device FILEN.
(3) Terminal symbol not in TRMLST.
(4) Missing parenthesis in a triplet.
(5) Triplet not completely contained in one input record.

A.2.3 INTERPRETER SPECIFICATION ROUTINES

(1) RECOGT (CODE,TRMLST,TSIZE)

Parameters

CODE: Mode Integer array
TRMLST: Mode Integer array
TSIZE: Mode Integer.

Function

To inform the interpreter of the addresses of the arrays containing the names of the lexical entries (TRMLST) and the code (CODE) to be used by the lexical recognizer. The size (TSIZE) of the table in TRMLST is also required. CODE, TRMLST, and TSIZE are to be in the format produced by the preprocessor TERMS.

(2) RECOGE (ERTABL,ESIZE)

Parameters

ERTABL: Mode Integer array

ESIZE: Mode Integer.

Function

To inform the interpreter of the address and size of the table of the error messages to be used. ERTABL, and ESIZE are to be in the format produced by the preprocessor ERRORS.

(3) RECOGC (COMMS,CSIZE)

Parameters

COMMS: Mode Integer array

CSIZE: Mode Integer.

Function

To inform the interpreter of the address and size of the table of the commands to be used. COMMS and CSIZE are to be in the format produced by the preprocessor COMAND.

Possible Error

The user defined module whose entry point is given in a triplet has not been loaded.

A.2.3 LABELLED COMMON

Three sets of labelled common are used: ATVALS, contains the attention variables that may be set by user actions. APARMS contains the arrays ACHAR, AX and AY, used as described for character, point or vector strings as specified in Table 8.1. SETING, contains four Boolean variables whose functions is described later in this Section.

(1) COMMON/ATVALS/ATCT,ATBN,ATID,ATTYPE,ATXP,ATYP,ATN, ATCODE,ATCTYP,ATFK,ATXA,ATYA,ATXB,ATYB,ATXC,ATYC, ATXRC,ATYRC,ATINT,ATREAL,ATS

(2) COMMON/APARMS/ACHRMS,AXYMAX,ACHAR,AX,AY

(3) COMMON/SETING/BUNCH,EOM,ERROR,DISCOM

These three sets of labelled common must be defined in at least one of the user-defined subroutines.

The use of four Boolean variables is as follows:

(1) BUNCH - If BUNCH is set to TRUE, the user may group more than one command in a single transmission from GRID. The commands are decoded in sequence and the appropriate program modules executed for each statement in turn. If BUNCH is set to FALSE only one command at a time may be transmitted.

(2) EOM - If EOM is set to TRUE, the end-of-message element in the sequence of user actions is to be decoded as a terminal symbol in the lexicon. For

this purpose, the attention variable ATCT takes on the value 6 for the end-of-message element. If EOM is set to FALSE, the element is disregarded when attempting to decode the user input.

(3) ERROR - The programmer may wish to display "execution-time error" messages on the CRT (Section 8.3.3). These messages may be defined in block 1. If ERROR is set to TRUE, the error message in block 1 will be displayed on the next transmission to GRID. It will be automatically deleted from the display file when the user returns control to the 360, and ERROR will be reset to FALSE.

(4) DISCOM - If DISCOM is set to TRUE, the sequence of user inputs for each transmission are processed by the lexical recognizer and listed on the logical device SPRINT. If DISCOM is set to FALSE, the listing is not given. This facility is useful when debugging, or attempting to monitor user actions for the purpose of duplicating a session at the terminal.

A.3 EXAMPLE

Let us consider the example given in Section 8.5, a set of six commands used in the Sieve Process. The entries required for the lexicon are:

DISPLAY ::= $atct=1\wedge atbn=2\wedge atid=1$ #

CREATE ::= $atct=1\wedge atbn=2\wedge atid=6$ #

CATENATE ::= $atct=1\wedge atbn=2\wedge atid=7$ #

WINDOW ::= $atct=1\wedge atbn=2\wedge atid=8$ #

RESTART ::= $atct=1\wedge atbn=2\wedge atid=9$ #

STOP ::= $atct=1\wedge atbn=2\wedge atid=10$ #

MAPNAME ::= $atct=1\wedge atbn\geq 40\wedge atbn\leq 60$ #

MAPNAME1 ::= $atct=5\wedge atn\leq 8$ #

LINE ::= $atct=4\wedge atn\leq 25\wedge atxa<850\wedge atya<850\wedge atxb<850\wedge atyb<850$ #

DIGIT ::= $atct=5\wedge atn=1\wedge atint\leq 8$ #

#

The error messages required are:

0 A COMMAND MUST BEGIN WITH ONE OF THE VERBS SPECIFIED ON THE RHS OF THE SCREEN

1 INVALID MAPNAME SELECTED

2 A MAPNAME MUST CONTAIN AT MOST EIGHT CHARACTERS

3 RESTART MUST APPEAR ALONE IN MESSAGE BUFFER

4 STOP MUST APPEAR ALONE IN MESSAGE BUFFER

5 DISPLAY COMMAND HAS ONLY 2 COMPONENTS

6 CREATE COMMAND HAS ONLY 2 COMPONENTS

7 INVALID LINE DATA INPUT

8 CATENATE COMMAND HAS ONLY 3 COMPONENTS

9 WINDOW COMMAND HAS ONLY 2 COMPONENTS

10 WINDOW NUMBERS RANGE FROM 0 TO 8

#

The command triplets are described as follows:

```
(DISPLAY 1 NULL) (MAPNAME 5 DISMAP) #

(CREATE 2 NULL) (MAPNAME1 6 CREATE) #

(CATENATE 1 NULL) (MAPNAME 7 NULL) (LINE 8 CATMAP) #

(WINDOW 10 NULL) (DIGIT 9 WINDOW) #

(RESTART 3 INITAL) #

(STOP 4 ENDJOB) #

#
```

Assume these are all keypunched on cards in the order presented above. The programmer may then write the following INITAL procedure.

```
      SUBROUTINE INITAL
      IMPLICIT INTEGER (A)
      COMMON/ATVALS/A(27)
      COMMON/APARMS/AC,AXY,ACHAR(25),AX(25),AY(25)
      COMMON/SETING/BUNCH,EOM,ERROR,DISCOM
      INTEGER*2 AX,AY,ACHAR
      LOGICAL*1 BUNCH,EOM,ERROR,DISCOM
      AXY=25
      AC=25
      DO 1 I = 1,27
1     A(I)=0
C         THIS INITIALIZES THE ATTENTION VARIABLES TO ZERO       C
      BUNCH=.FALSE.
      EOM=.FALSE.
      ERROR=.FALSE.
      DISCOM=.TRUE.
C         THIS INITIALIZES THE VARIABLES IN SETING               C
C         THIS ALLOWS FOR THE USER INPUTS TO BE DISPLAYED        C
C         ON SPRINT (AN MTS OUTPUT FILE)                         C
C         THE REST OF THE ROUTINE INITIALIZES THE TABLES         C
C         FOR THE INTERPRETER AND THE DISPLAY FILE               C
      DIMENSION ICODE (200),ITERMS(30),ICOMMS (50),IERS(100)
      CALLTERMS (ICODE,ITERMS,5,IT,200,30,&10)
      CALL ERRORS (IERS,5,IE,100,&10)
      CALL COMAND (ICOMMS,5,IC,ITERMS,IT,50,&10)
C         THE DATA IS ON UNIT 5                                  C
C         AT THIS POINT THE DATA IN ARRAYS IERS, ICOMMS,         C
C         ICODE AND ITERMS MAY BE PRESERVED ON DISC FOR          C
C         USE IN OTHER SESSIONS                                  C
      CALL RECOGE (IERS,IE)
      CALL RECOGC (ICOMMS,IC)
      CALL RECOGT (ICODE,ITERMS,IT)
C         THE ABOVE THREE CALLS INITIALIZE THE INTERPRETER     C
C         TABLES THEY MAY BE IN ANY ORDER                      C
C         AT THIS POINT IN THE PROGRAM THE USER DEFINES        C
C         THE DISPLAY FILE AS HE WISHES IT TO APPEAR ON        C
C         THE FIRST TRANSMISSION TO GRID                       C
      RETURN
10    STOP
C         ERROR HALT - BEFORE STOPPING THE USER MAY PRINT      C
C         OUT DATA IN ORDER TO CORRECT THE ERROR               C
      END
```

The other procedures used in the command specification would also be supplied when attempting to run the program.

If desired, the user may call TERMS, ERRORS and COMAND from a main program and preserve the resulting data on disc. For any session at the CRT the data may then be read from disc and passed to the interpreter. This option allows the user to "debug", to an extent, the tabular data for the interpreter before using the CRT.